НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

До захисту допущено

Завідувач кафедри

_____ Ігор ПАРХОМЕЙ

«___»_____2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інтелектуальних та робототехнічних систем»
спеціальність 121 «Інженерія програмного забезпечення»

на тему: «Структурно-параметричний синтез гібридних нейронних мереж ансамблевої топології»

Виконав:
студент IV курсу, групи ІТ-62
Рязановський Кирило Денисович _____

Керівник:
доцент, к.т.н., доц.,
Чумаченко Олена Іллівна _____

Консультант з норм. контролю:
доцент, к.т.н, доц.
Пасько Віктор Петрович _____

Рецензент:
старший викладач, к.т.н., ст. викл.,
Василенко Микола Павлович _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2020 року

Пояснювальна записка
до дипломного проєкту
на тему: «Структурно-параметричний синтез гібридних нейронних мереж ансамблевої топології»

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра технічної кібернетики

Рівень вищої освіти – перший (бакалаврський)

Спеціальність 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення інтелектуальних та робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Ігор ПАРХОМЕЙ

«___»_____2020 р.

ЗАВДАННЯ

на дипломний проєкт студенту

Рязановському Кирилу Денисовичу

1. Тема проєкту «Структурно-параметричний синтез гібридних нейронних мереж ансамблевої топології», керівник проєкту Чумаченко Олена Іллівна, доцент, кандидат технічних наук, затверджені наказом по університету від «07» травня 2020р. № 1081-с

2. Термін подання студентом проєкту: 23.05.2020 р.

3. Вихідні дані до проєкту: ансамблеві алгоритми синтезу нейронних мереж, системи прогнозування реального часу, методи побудови гібридних нейронних мереж.

4. Зміст пояснювальної записки:

1. Аналіз існуючих алгоритмів гібридизації і синтезу нейронних мереж;
2. Розробка та дослідження власних методів побудови моделей машинного навчання ансамблевої топології;
3. Створення прикладного програмного пакету та інтерфейсів доступу до розроблених алгоритмів та моделей;
4. Реалізація системи прогнозування реального часу роботехнічного модуля.

Висновки до роботи.

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій тощо): Частина архітектури скороченого ансамблю (А3), Частина архітектури мета-дерева (А3), Діаграма послідовності створення ансамбля (А3), Діаграма послідовності для прогнозування ансамблем (А3).

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Перевірка на співпадіння	доц. Лісовиченко О.І.		
Норм. Контроль	доц. Пасько В.П.		

7. Дата видачі завдання «01» жовтня 2019р.

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1	Аналіз існуючих алгоритмів гібридизації і синтезу нейронних мереж.	01.10.19-31.10.19	
2	Розробка та дослідження власних структур та методів побудови моделей машинного навчання ансамблевої топології.	01.11.19-10.01.19	
3	Розробка прикладного програмного пакету та інтерфейсів доступу до розроблених алгоритмів і моделей нейронних мереж	11.01.19-18.03.20	
4	Реалізація контрольних прикладів системи.	19.03.20-30.05.20	
5	Оформлення дипломної роботи.	01.06.20-08.06.20	
6	Попередній захист.	09.06.20	
7	Захист.		

Студент Кирило РЯЗАНОВСЬКИЙ

Керівник проєкту Олена ЧУМАЧЕНКО

АНОТАЦІЯ

Дипломна робота виконана на 107 сторінках і містить 39 ілюстрацій, 11 додатків. При розробці використано інформацію з 46 джерел.

Метою даної роботи є наукове дослідження та подальша практична розробка і програмна реалізація алгоритмів і моделей структурно-параметричного синтезу гібридних нейронних мереж на основі ансамблевої топології у вигляді окремого програмного пакету з інтерфейсом доступу.

У роботі проаналізовані сучасні підходи до гібридизації та синтезу нейронних мереж, розглянуті та досліджені проблеми побудови та підтримки таких структур. Були запропоновані альтернативні методи для створення та управління ансамблевих архітектур.

Розроблені методи були реалізовані практично, перевірені та порівняні з існуючими при вирішенні різних класів задач. Показана перевага запропонованих підходів та можлива інтеграція з розглянутими структурами.

Був розроблений програмний пакет з програмним інтерфейсом створення, керування, імпорту та експорту моделей машинного навчання в ансамблевих структурах. Розроблене програмне забезпечення уніфікує інтерфейс моделі і дозволяє абстрагуватися від практичної реалізації нейронної мережі при побудові ансамблів мереж.

Розроблений пакет був застосований для побудови прототипу системи прогнозування реального часу інтелектуального модуля робототехнічної системи. Система дозволяє швидко та точно продукувати наступний рух робота на основі багатьох датчиків. Модель підтримує баланс розміру, швидкості і точності, тому прототип готовий до інтеграції в реальну систему керування.

Ключові слова: гібридна нейронна мережа, ансамблева топологія, структурно-параметричний синтез, машинне навчання, система реального часу.

SUMMARY

Research paper performed at 107 pages and contains 39 illustrations, 11 appendices. Information from 46 sources was used in the development.

The purpose of this work is research and further practical development and software implementation of algorithms and models of structural-parametric synthesis of hybrid neural networks based on ensemble topology in the form of a separate software package with access interface.

The modern approaches to hybridization and synthesis of neural networks are analyzed in the work, the problems of construction and maintenance of such structures are considered and investigated. Alternative methods for creating and managing ensemble architectures have been proposed.

The developed methods were implemented in practice, tested and compared with existing ones in solving different classes of problems. The advantage of the proposed approaches and possible integration with the considered structures is shown.

A software package with a programming interface for creating, managing, importing and exporting machine learning models in ensemble structures was developed. The developed software unifies the model interface and allows to abstract from the practical implementation of the neural network in the construction of network ensembles.

The developed package was used to build a prototype of a real-time prediction system for an intelligent module of a robotic system. The system allows to quickly and accurately produce the next robot movement based on many sensors. The model maintains a balance of size, speed and accuracy, so the prototype is ready for integration into a real control system.

Keywords: hybrid neural network, ensemble topology, structural-parametric synthesis, machine learning, real-time system.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ШІ – штучний інтелект

НМ – нейронна мережа

ГНМ – гібридна нейронна мережа

МГУА – метод групового урахування аргументів

LSTM – long-short term memory

GRU – gated recurrent unit

MAPE – mean absolute percentage error

MSE – mean squared error

SGD – stochastic gradient descent

ІВ – індивідуальний внесок

TSK – Takagi-Sugeno-Kang inference system

CNN – convolutional neural network

ПЗ – програмне забезпечення

ООП – об’єктно-орієнтоване програмування

API – application programming interface

ВСТУП

Завдяки розвитку Інтернету, технічних можливостей сучасних апаратних засобів кількість даних, які отримуються і генеруються, в день обчислюється вже пета-, а то і ексабайтами. Крім зберігання такої величезної кількості неструктурованої інформації, потрібна ще й обробка, аналіз, розумне уявлення, структурування цього обсягу для отримання нових знань (про клієнтів, виробництві і т.д.), прогнозування (кількості товарів, вартості акцій, температурний режим і т. д.), прийняття рішень (видавати кредит особі чи ні і т.д.) і т.п.

Використання методів ШІ і машинного навчання в вирішенні складних прикладних задач аналізу різних даних неухильно росте вгору. Причиною тому служить велика гнучкість, масштабованість і толерантність даних методів. Доказом тому служить активний розвиток департаментів ШІ в найбільших світових корпораціях, наприклад, DeepMind в Google [1], AWS AI в Amazon [2], Microsoft Research Lab – AI в Microsoft [3]. Дані організації активно розвивають, впроваджують і популяризують сферу ШІ в світі, що тільки підтверджує актуальність.

Одним з основних інструментів машинного навчання є штучні нейронні мережі. Існує величезна кількість різних архітектур таких мереж для вирішення самих нетривіальних завдань. За останні кілька років використання НМ для обробки даних стало досить ефективним і популярним [4, 5]. Це не дивно, адже вони є універсальними аппроксиматорами, які можна використовувати в різних сферах: від класифікації зображень до підтримки прийняття рішень.

Використання ШНМ дозволяє ефективно справлятися з проблемами, які раніше були непідвладні іншим більш простим моделям. Їх перевага перед класичними алгоритмами полягає в можливості динамічної настройки і зміні структури мережі, її параметрів.

Великим стрибком у розвитку нейронних мереж стало глибоке навчання (deep learning), завдяки якому сучасні інструменти перекладу тексту, розпізнавання мови, комп'ютерного зору, автоматичного водіння автомобіля і інші стали досить точними і загальнодоступними.

Запропоновано велику кількість різних специфічних архітектур мереж [6], які краще пристосовуються для вирішення певних завдань, але в той же час мають ряд обмежень і недоліків.

Однак створення і навчання однієї великої глибокої НМ має безліч пов'язаних проблем: перенавчання, налаштування і оптимізація гіперпараметрів, потрібно високопродуктивне обладнання і т.д. Тому інший гілкою розвитку НМ і глибокого навчання стало використання модульних топологій: одна велика глибока мережа, яка складається з декількох невеликих базових мереж, кожна з яких може виконувати окрему функцію (наприклад, кластеризація, зменшення розмірності даних, уточнення результатів, як і, власне, саме рішення поставленого завдання)

Надалі для збільшення точності відбувається об'єднання таких модулів в ансамблевий структуру. Така зв'язка дозволяє компенсувати недоліки однієї структури перевагами іншої, що є неможливим при використанні тільки однієї мережі і є безумовною перевагою.

РОЗДІЛ 1 ШТУЧНІ НЕЙРОННІ МЕРЕЖІ. КЛАСИФІКАЦІЯ. АНАЛІЗ

1.1 Область застосування штучних нейронних мереж

Як вже було сказано, технології НМ є універсальними для вирішення безлічі різних завдань. Динамічність, гнучкість і різноманітність архітектур дозволяє використовувати дані структури в різних сферах: інженерії, бізнесі, медицині, сільському господарстві, військовій сфері, сфері захисту громадян та інші.

Наприклад, нижче наведений список використання нейронних мереж в інженерії [7].

- Авіація і космонавтика: детектори і моделювання несправностей авіаційних компонентів, системи управління літаком, високопродуктивне автопілотування і моделювання траєкторії польоту.
- Автомобілі: вдосконалені системи наведення, розробка силових агрегатів, віртуальних датчиків і аналізаторів гарантійної активності.
- Електроніка: аналіз відмов мікросхем, схеми мікросхем, машинне зір, нелінійне моделювання, передбачення кодової послідовності, управління процесом і синтез мови.
- Виробництво: аналіз конструкції хімічної продукції, динамічне моделювання систем хімічних процесів, управління процесом, діагностика процесів і машин, проєктування і аналіз продукції, прогнозування якості паперу, визначення тендерів, планування і управління проєктами, аналіз якості комп'ютерних чіпів, системи візуального контролю якості та аналіз якості зварювання.
- Механіка: моніторинг стану, моделювання систем і контроль.
- Робототехніка: автонавантажувачі, маніпулятори, системи контролю траєкторії і системи спостереження.

– Телекомунікації: управління мережею банкоматів, автоматизовані інформаційні послуги, системи обробки платежів клієнтів, стиснення даних, еквалайзери, управління помилками, розпізнавання рукописного введення, проєктування, управління, маршрутизація і контроль, моніторинг мережі, переклад розмовної мови в режимі реального часу і розпізнавання образів (особи, об'єкти, відбитки пальців, семантичний аналіз, перевірка орфографії, обробка сигналів і розпізнавання мови).

Нижче наведені поточні приклади використання НМ.

– Банківська справа: випуск і управління кредитними картами, оцінка кредитних і депозитних заявок, шахрайство і оцінка ризиків, а також прострочені кредити.
– Бізнес-аналітика: моделювання поведінки клієнтів, сегментація клієнтів, схильність до шахрайства, дослідження ринку, структура ринку і моделі виснаження, дефолту, покупки і продовження.
– Захист: боротьба з тероризмом, розпізнавання осіб, витяг ознак, придушення шуму, розпізнавання об'єктів, датчики, сонар, обробка сигналів радара і зображення, ідентифікація сигналу/зображення, відстеження цілі і управління зброєю.
– Освіта: програмне забезпечення для адаптивного навчання, динамічне прогнозування, аналіз і прогнозування системи освіти, моделювання успішності учнів і профілювання особистості.
– Фінансовий: рейтинги корпоративних облігацій, аналіз корпоративних фінансів, аналіз використання кредитних ліній, прогнозування валютних цін, консультування по кредитах, перевірка іпотечних кредитів, оцінка нерухомості і торгівля портфелями.
– Медицина: аналіз ракових клітин, аналіз ЕКГ і ЕЕГ, консультування з питань невідкладної допомоги, зниження витрат і поліпшення якості лікарняних систем, оптимізація процесу трансплантації і проєктування протеза,

використання згорткових нейронних мереж для аналізу рентгенів, МРТ та інших зображень для визначення хвороби.

– Цінні папери: автоматичний рейтинг облігацій, аналіз ринку і системи торгівлі акціями
– Транспорт: системи маршрутизації, системи діагностики гальм вантажних автомобілів і складання розкладу руху транспортних засобів.

Як можна помітити, ШІ, а зокрема НМ, проникають практично в усі сфери нашого життя. Список можливих застосувань і областей є дуже великим і до того ж постійно розширюється. З кожним роком кількість дослідників стає все більше, їх фантазія і уява знайдуть ще більше застосувань

1.2 Структура штучного нейрона. Функції активації

1.2.1 Історична довідка

Перш ніж обговорювати персептрон і пов'язані з ним алгоритми більш детально, слід коротко розглянути перші етапи машинного навчання. Намагаючись зрозуміти, як працює біологічний мозок, Уоррен Маккаллок і Уолтер Піттс в 1943 році опублікували першу концепцію спрощеної мозкової клітини, так званого нейрона Маккаллок-Питтса (MCP), в 1943 році. Нейрони являють собою взаємопов'язані нервові клітини в головному мозку, які беруть участь в обробці і передачі хімічних і електричних сигналів (рис. 1.1).

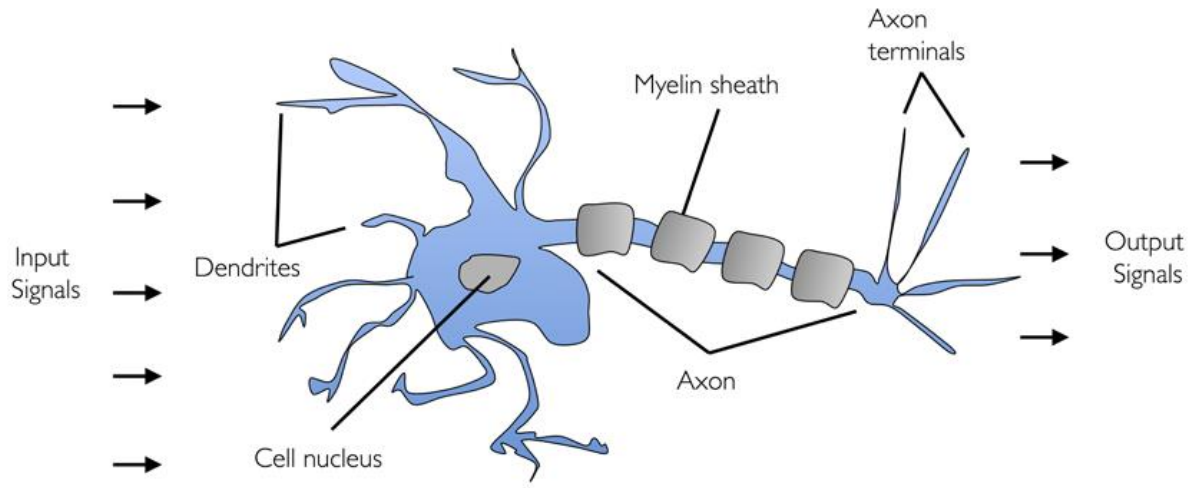


Рис. 1.1 Структура нейрону [8]

Маккалок і Піттс описали таку нервову клітину як простий логічний елемент з двійковими виходами; кілька сигналів надходять на дендрити, потім інтегруються в тіло клітини, і, якщо накопичений сигнал перевищує певний поріг, генерується вихідний сигнал, який буде передаватися аксоном.

Всього кілька років тому Френк Розенблат опублікував першу концепцію правила навчання персептрона, засновану на моделі нейронів MCP. З його правилом персептрона Розенблатт запропонував алгоритм, який автоматично навчав би оптимальні вагові коефіцієнти, які потім множилися на вхідні дані, щоб прийняти рішення, активується чи нейрон чи ні. У контексті навчання з учителем і класифікації такий алгоритм може потім використовуватися для прогнозування того, чи належить зразок одного класу або іншому.

1.2.2 Формальне визначення штучного нейрона

Більш формально, ми можемо висвітлити ідею, що стоїть за штучними нейронами, в контекст завдання двійковій класифікації, де для простоти ми називаємо наші два класи 1 (позитивний клас) і -1 (негативний клас). Потім ми можемо визначити процедуру прийняття рішення ($\varphi(z)$), яка приймає лінійну

комбінацію певних вхідних значень $\boldsymbol{x}$ і відповідного вагового вектору $\boldsymbol{w}$, де z – так званий чистий вхід нейрона:

$$z = \boldsymbol{w}^T \boldsymbol{x}, \boldsymbol{w} = \begin{bmatrix} w_1 \\ \dots \\ w_m \end{bmatrix}, \boldsymbol{x} = \begin{bmatrix} x_1 \\ \dots \\ x_m \end{bmatrix}. \quad (1.1)$$

Тепер, якщо вхідний сигнал конкретної вибірки $\boldsymbol{x}^{(i)}$ перевищує певний поріг θ, ми прогнозуємо клас 1 і клас -1 в іншому випадку. В алгоритмі персептрона вирішальна функція $\varphi(.)$ є варіантом функції з одиничним кроком:

$$\varphi(z) = \begin{cases} 1, z \geq \theta, \\ -1, z < \theta \end{cases}. \quad (1.2)$$

Для простоти поріг переноситься в ліву частину рівняння і визначається нульову вагу як $w_0 = -\theta$ и $x_0 = 1$, щоб записати z в більш компактній формі:

$$z = w_0 x_0 + w_1 x_1 + \dots + w_m x_m = \boldsymbol{w}^T \boldsymbol{x}, \quad (1.3)$$

$$\varphi(z) = \begin{cases} 1, z \geq 0, \\ -1, z < 0 \end{cases}. \quad (1.4)$$

В літературі по машинному навчання негативний поріг, або вага $w_0 = -\theta$ зазвичай називається зміщенням.

На зображенні (рис. 1.2) показано, як вхідний сигнал $z = \boldsymbol{w}^T \boldsymbol{x}$ перетвориться в двійковий вихід (-1 або 1) за допомогою функції активації персептрона (ліва подфігура) і як його можна використовувати для розрізнення двох лінійно поділюваних класів (права подфігура):

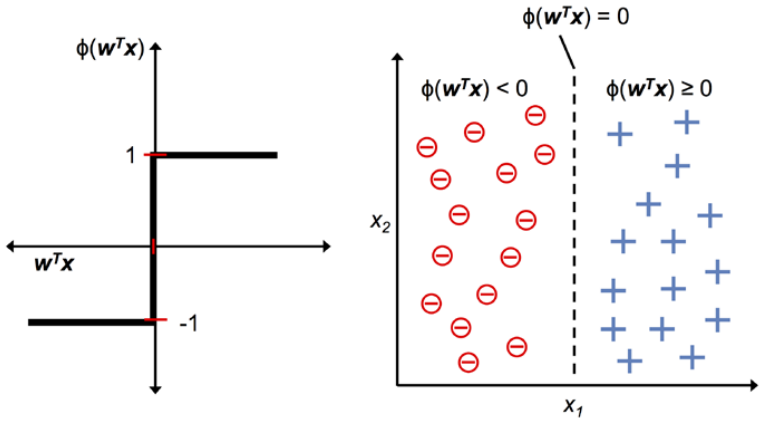


Рис. 1.2 Приклад роботи персептрона Розенблатта [8]

Персептрон отримує вхідні дані вибірки ***x*** і об'єднує їх з вагами ***w*** для обчислення нового входу. Потім вхідний сигнал передається в порогову функцію, яка генерує бінарний вихідний сигнал -1 або +1 – передбачену мітку класу зразка. На етапі навчання цей висновок використовується для розрахунку помилки прогнозу і поновлення ваг (рис. 1.3).

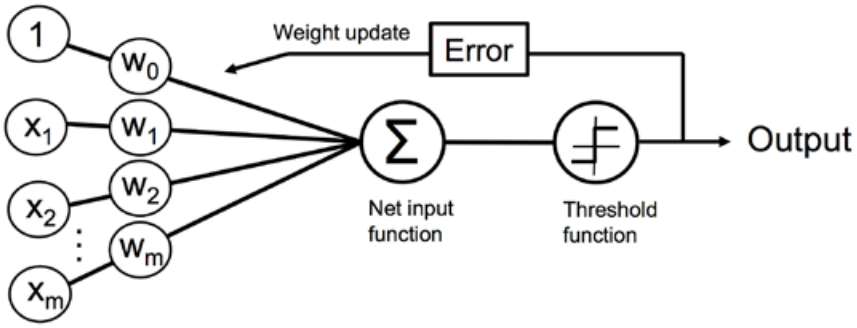


Рис. 1.3 Структура персептрона [8]

1.2.3 Функції активації

У наш час від базової конструкції персептрону, звичайно ж, ніхто не відмовляється. Сенс і основна послідовність і раніше ті ж самі: спочатку в персептрон надходять входи з даних або попередніх рівнів мережі, потім береться їх лінійна комбінація з деякими вагами, які, власне, і будуть навчатися в мережі, а потім результат проходить через деяку нелінійну функцію , без якої, як ми вже бачили в цьому розділі, ніякої виразної сили у нейронної мережі не вийде.

Саме з таких перцептронів, або нейронів, складаються всі сучасні нейронні мережі. Різниця є лише в тому, яка, власне, конструкція нелінійності.

Історично в нелінійних перцептронах зазвичай застосовувалася функція активації (збудження нейрона) у вигляді логістичного сигмоїда: $\sigma(x)=\frac{1}{1+e^{-x}}$. Але є багато різних функцій активації, які в різний час і для різних цілей використовувалися в літературі (табл. 1.1) [9].

Таблиця 1.1. Різні функції активації: зведена таблиця

Назва функції	Формула *f(x)*	Похідна *f'(x)*
Логістичний сигмоїд σ	$\frac{1}{1+e^{-x}}$	$f(x)(1-f(x))$
Гіперболічний тангенс	$\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$1-f^{2}(x)$
SoftSign	$\frac{x}{1+\lvert x\rvert}$	$\frac{1}{(1+\lvert x\rvert)^{2}}$
Функція Хевісайда	$\begin{cases}0, x<0,\\1, x\geq 0\end{cases}$	0
SoftPlus	$\log(1+e^{x})$	$\frac{1}{1+e^{-x}}$
ReLU	$\begin{cases}0, x<0,\\x, x\geq 0\end{cases}$	$\begin{cases}0, x<0,\\1, x\geq 0\end{cases}$
Leaky ReLU, Parameterized ReLU	$\begin{cases}ax, x<0,\\x, x\geq 0\end{cases}$	$\begin{cases}a, x<0,\\1, x\geq 0\end{cases}$
ELU	$\begin{cases}a(e^{x}-1), x<0,\\x, x\geq 0\end{cases}$	$\begin{cases}f(x)+a, x<0,\\1, x\geq 0\end{cases}$

Насправді архітектура системи і алгоритми оптимізації зазвичай набагато важливіше. У більшості випадків використовується одну з двох функцій активації: або логістичний сигмоїд, або ReLU [9].

1.3 Класифікація нейронних мереж

Нейронні мережі – це набори алгоритмів, призначених для розпізнавання шаблонів і інтерпретації даних за допомогою кластеризації або маркування. Іншими словами, нейронні мережі є алгоритмами. Алгоритм навчання – це метод, який використовується для виконання процесу навчання нейронної мережі. Оскільки існує величезна кількість доступних алгоритмів та архітектур навчання, кожен з яких складається з різних характеристик і продуктивності, використовуються різні алгоритми для досягнення різних цілей.

Нижче наведені основні на даний момент архітектури НМ (табл. 1.2) [7].

Таблиця 1.2. Архітектури нейронних мереж

Архітектура/Алгоритм	Призначення
Autoencoder (AE)	Зазвичай AE використовується, щоб зменшити кількість розглянутих випадкових величин, щоб система могла вивчити уявлення для набору даних і, отже, обробляти генеративні моделі даних
Bidirectional Recurrent Neural Network (BRNN)	Мета BRNN – збільшити кількість інформаційних входів, доступних для мережі, підключивши два прихованих, спрямованих протилежно один одному шару до одного і того ж виходу
Boltzmann Machine (BM)	Рекурентна нейронна мережа, цей алгоритм здатний вивчати внутрішні представлення даних і може представляти і вирішувати складні комбіновані завдання
Convolutional Neural Network (CNN)	CNN, найбільш часто використовувані для аналізу візуальних образів, є нейронну мережу з прямим зв'язком, призначену для мінімізації попередньої обробки
Deep Belief Network (DBN)	DBN може навчитися відновлювати свої входи імовірнісно, використовуючи шари в якості детекторів ознак
Deep Convolutional Inverse Graphics Network (DCIGN)	Модель DCIGN спрямована на вивчення інтерпретується представлення зображень, які система розділяє відповідно до елементів тривимірної структури сцени, такими як зміни освітлення і повороти глибини. DCIGN використовує багато рівнів операторів, як згорткових, так і дезгорткових

Таблиця 1.2. (продовження)

Deep Residual Network (DRN)	DRN допомагають у вирішенні складних завдань і моделей глибокого навчання. Маючи багато шарів, DRN запобігає погіршенню результатів
Denoising Autoencoder (DAE)	DAE використовується для відновлення даних з пошкоджених вхідних даних; алгоритм змушує прихований шар знаходити більш надійні ознаки. В результаті вихідні дані дають більш вдосконалену версію вхідних даних
Echo State Network (ESN)	ESN працює з випадковою великий фіксованою рекурентною, нейронною мережею, в якій кожен вузол отримує нелінійний відповідний сигнал. Алгоритм випадковим чином встановлює і призначає ваги і зв'язність для досягнення гнучкості навчання
Extreme Learning Machine (ELM)	Цей алгоритм навчає вихідні значення ваг прихованих вузлів за один крок, створюючи лінійну модель. ELM можуть добре узагальнювати і вчитися у багато разів швидше, ніж мережі зворотного поширення
Feed Forward Neural Network (FF or FFNN) and Perceptron (P)	Це основні алгоритми для нейронних мереж. Нейронна мережа з прямим зв'язком – це штучна нейронна мережа, в якій з'єднання вузлів не утворюють цикл; персептрон – це бінарна функція з двома результатами (вгору / вниз; так / ні, 0/1)
Gated Recurrent Unit (GRU)	GRU використовують з'єднання через послідовності вузлів для виконання завдань машинного навчання, пов'язаних з кластеризацією і пам'яттю. GRU уточнюють результати за допомогою управління інформаційним потоком моделі
Generative Adversarial Network (GAN)	Ця система протиставляє дві нейронні мережі – дискримінаційну і породжує. Мета полягає в тому, щоб розрізняти реальні та синтетичні результати, щоб імітувати концептуальні завдання високого рівня
Hopfield Network (HN)	Ця форма рекурентної штучної нейронної мережі являє собою асоціативну систему пам'яті з двійковими граничними вузлами. Призначені для досягнення локального мінімуму, HN надають модель для розуміння людської пам'яті

Таблиця 1.2. (продовження)

Kohonen Network (KN)	KN організовує проблемне простір в двовимірне відображення. Різниця між самоорганізації картами (SOM) і іншими підходами до вирішення проблем полягає в тому, що SOM використовують конкурентну навчання, а не навчання з виправленням помилок
Liquid State Machine (LSM)	Відомий як машинне навчання третього покоління (або сплеску нейронної мережі), LSM додає концепцію часу як елемента. LSM генерують активацію просторово-часової нейронної мережі, оскільки вони зберігають пам'ять під час обробки. В фізики і обчислювальна нейробіології використовуються LSM
Long/Short-Term Memory (LSTM)	LSTM здатний вивчати або запам'ятовувати порядок в задачах прогнозування, що стосуються послідовності. Модуль LSTM містить осередок, вхідний шлюз, вихідний шлюз і шлюз забування. Осередки зберігають значення протягом довільних тимчасових інтервалів. Кожен блок регулює потоки значень через з'єднання LSTM. Ця можливість упорядкування важлива в складних проблемних областях, таких як розпізнавання мови і машинний переклад
Markov Chain (MC)	MC – це математичний процес, який описує послідовність можливих подій, в яких імовірність кожної події залежить виключно від стану, досягнутого в попередньому подію. Приклади використання включають в себе передбачення слів при наборі і Google PageRank
Neural Turing Machine (NTM)	Заснований на роботах вченого по обробці даних Алана Тюрінга в середині 20-го століття, NTM виконує обчислення і розширює можливості нейронних мереж, зв'язуючись із зовнішньою пам'яттю. Розробники використовують NTM в роботах і розглядають його як один із засобів для створення штучного людського мозку

Таблиця 1.2. (закінчення)

Radial Basis Function Networks (RBF nets)	Розробники використовують мережі RBF для моделювання даних, що представляють основний тренд або функцію. Мережі RBF вчаться наближати основний тренд, використовуючи дзвоновидні криві або нелінійні класифікатори. Нелінійні класифікатори аналізують більш глибоко, ніж прості лінійні класифікатори, які працюють з векторами меншого розміру. Використовуються ці мережі для управління системою і прогнозування часових рядів
Recurrent Neural Network (RNN)	RNNs моделюють послідовні взаємодії через пам'ять. На кожному часовому кроці RNN обчислює нову пам'ять або приховане стан, залежне як від поточного входу, так і від попереднього стану пам'яті. Програми включають музичну композицію, управління роботом і розпізнавання дій людини
Support Vector Machine (SVM)	На основі навчальних наборів прикладів, які відносяться до однієї з двох можливих категорій, алгоритм SVM створює модель, яка присвоює нові приклади одного з двох класів. Потім модель являє приклади у вигляді відображених точок в просторі, в той час як ці окремі приклади категорій діляться по максимально широкому розриву між ними. Потім алгоритм відображає нові приклади в тому ж просторі і пророкує, до якої категорії вони належать, виходячи з того, яку сторону розриву вони займають. Програми включають в себе визначення обличчя та біоінформатику
Variational Autoencoder (VAE)	VAE – це особливий тип нейронної мережі, який допомагає створювати складні моделі на основі наборів даних. Загалом, автоенкодер – це мережа глибокого навчання, яка намагається реконструювати модель або зіставити цільові вихідні дані з наданими вхідними даними за допомогою зворотного поширення. VAE також дає найсучасніші результати машинного навчання в областях формування зображень і навчання з підкріпленням

Деякі з представлених архітектур дуже гарно підходять для структурно-параметричного синтезу. Зокрема, мережа Кохонена, автоенкодер або варіаційний автоенкодер гарно використовуються при побудові модульної топології ГНМ. Вони можуть робити предобробку даних для подальшої передачі предоброблених даних на базову мережу, в якості якої може використовуватися багато інших різних мереж, наприклад, CNN, LSTM, GRU, SVM, FFNN.

1.4 Методи машинного навчання

Існує три типи машинного навчання (рис. 1.4):

– контрольоване навчання (supervised learning);
– неконтрольоване навчання (unsupervised learning);
– навчання з підкріпленням (reinforcement learning).

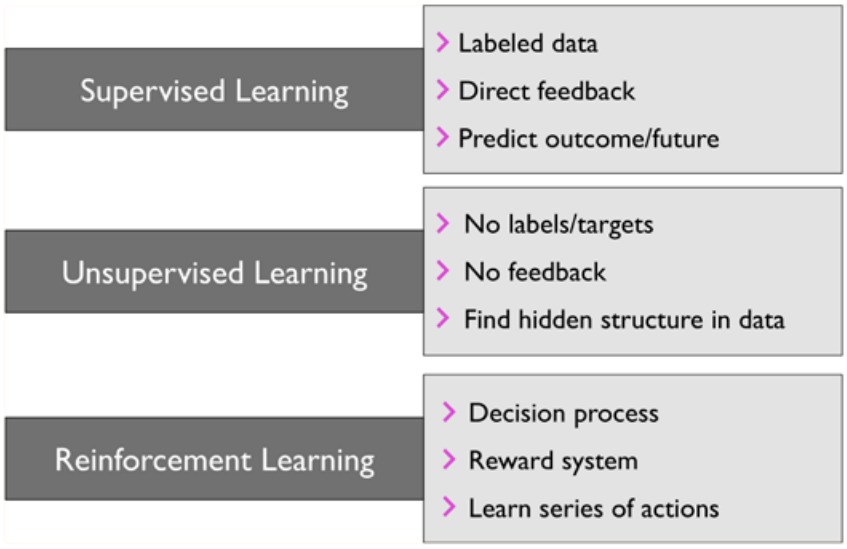


Рис. 1.4 Три типи машинного навчання [8]

1.4.1 Контрольоване навчання

Основна мета контрольованого навчання або навчання з учителем полягає в тому, щоб вивчити модель на основі розмічених систему адаптації, яка дозволяє нам робити прогнози щодо ще небачених або майбутніх даних. Тут термін

«контрольований» відноситься до набору вибірок, де бажані вихідні сигнали (мітки) вже відомі.

Розглядаючи приклад фільтрації спаму в електронній пошті, ми можемо навчити модель, використовуючи контрольований алгоритм машинного навчання, на корпусі відмічених листів, які правильно позначені як спам або не спам, щоб передбачити, чи належить новий лист однієї з двох категорії. Завдання навчання з учителем з дискретними мітками класів, як, наприклад, в попередньому прикладі фільтрації спаму в електронній пошті, також називається завданням класифікації. Інший підкатегорією контрольованого навчання є регресія, де сигнал результату являє собою безперервне значення. [8]

1.4.2 Навчання з підкріпленням

Інший тип машинного навчання – навчання з підкріпленням. У навчанні з підкріпленням мета полягає в тому, щоб розробити систему (агента), яка покращує свою продуктивність на основі взаємодії з навколишнім середовищем. Оскільки інформація про поточний стан середовища зазвичай також включає в себе так званий сигнал винагороди, ми можемо розглядати навчання з підкріпленням як область, пов'язану з контрольованим навчанням. Проте, в навчанні з підкріпленням ця зворотний зв'язок не є правильною міткою або значенням істинного підстави, а мірою того, наскільки добре дія була виміряна функцією винагороди. За допомогою взаємодії з навколишнім середовищем агент може потім використовувати навчання з підкріпленням для вивчення серії дій, які максимізують цю нагороду за допомогою дослідницького методу проб і помилок або обдуманого планування.

Популярним прикладом навчання підкріпленню є шаховий движок. Тут агент вибирає серію ходів в залежності від стану ігрового поля (оточення), і нагорода може бути визначена як перемога чи поразка в кінці гри.

Є багато різних підтипів навчання з підкріпленням. Проте, загальна схема полягає в тому, що агент в навчанні підкріплення намагається максимізувати винагороду шляхом серії взаємодій з навколишнім середовищем. Кожне стан може бути пов'язано з позитивним або негативним винагородою, і винагорода може бути визначено як досягнення спільної мети, такої як виграш чи програш у грі в шахи. Наприклад, в шахах результат кожного кроку можна уявити, як інший стан навколишнього середовища.

Для подальшого вивчення шахового прикладу візьмемо на увазі, що відвідування певних місць на шахівниці пов'язано з позитивною подією, наприклад, з видаленням шахової фігури супротивника з дошки або загрозою королеві. Інші позиції, однак, пов'язані з негативною подією, таким як втрата шахової фігури противнику в наступному ході. Тепер не кожен хід призводить до видалення шахової фігури, а навчання з підкріпленням пов'язано з навчанням серії кроків шляхом максимізації винагороди, заснованого на негайній і затриманої зворотного зв'язку. [8]

1.4.3 Неконтрольоване навчання

У контрольованому навчанні ми заздалегідь знаємо правильну відповідь при навчанні нашої моделі, а в навчанні з підкріпленням ми визначаємо міру винагороди за конкретні дії агента. Однак при навчанні без учителя ми маємо справу з немаркованими даними або даними невідомої структури. Використовуючи неконтрольовані методи навчання, ми можемо досліджувати структуру наших даних для вилучення значущої інформації без вказівки відомої змінної результату або функції винагороди. [8]

1.5 Методи навчання нейронних мереж

Завдання навчання формулюється з точки зору мінімізації функції втрат, *f*. Це функція, яка вимірює продуктивність нейронної мережі на наборі даних.

Функція втрат, як правило, складається з помилки і членів регуляризації. Термін помилки оцінює, як нейронна мережа описує набір даних. Термін регуляризації використовується для запобігання перенавчання шляхом контролю ефективної складності нейронної мережі.

Функція втрат залежить від адаптивних параметрів (зміщення і синоптичних ваг) в нейронної мережі. Вони зручно групуються в один *n*-мірний ваговій вектор ***w***.

На зображенні (рис. 1.5) наведений простий приклад функції втрат ***f(w)***.

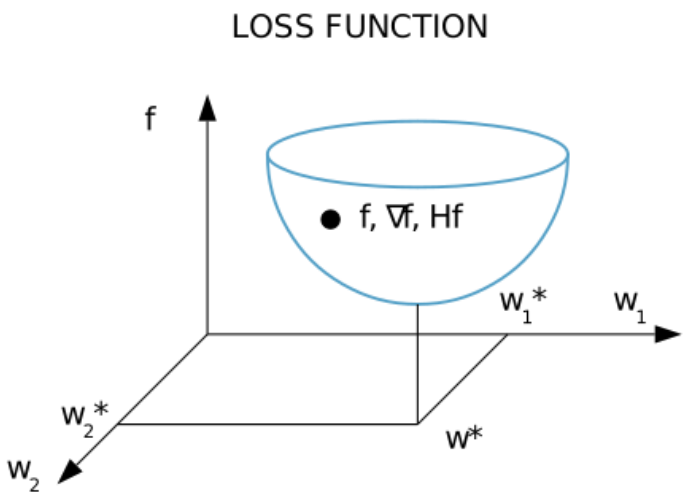


Рис. 1.5 Приклад функції втрат *f*

Як можна побачити, мінімум функції втрат відбувається в точці $\boldsymbol{w}^*$.

Завдання навчання для нейронних мереж формулюється як пошук вектору параметрів ***w****, при якому функція втрат *f* приймає мінімальне значення.

Функція втрат, як правило, є нелінійної функцією параметрів. Як наслідок, неможливо знайти найкращий алгоритм навчання для знаходження мінімуму. Замість цього ми розглядаємо пошук в просторі параметрів, що складається з послідовності кроків. На кожному етапі втрати будуть зменшуватися шляхом налаштування параметрів нейронної мережі.

Таким чином, для навчання нейронної мережі ми починаємо з деякого вектору параметрів (часто обирається випадковим чином). Потім ми генеруємо

порядок опцій, так що функція втрат зменшується на кожній ітерації алгоритму. Зміна втрат між двома кроками називається зменшенням втрат. Алгоритм навчання зупиняється, коли виконується зазначена умова або критерій зупинки. [10]

Існує два основні підходи до навчання НМ, а також їх різні модифікації. Розглянемо кожен з них.

1.5.1 Градієнтні методи [12]

Метод найшвидшого спуск є найпростішим алгоритмом навчання. Він вимагає інформації від вектору градієнта, і, отже, це метод першого порядку.

Позначимо $\Delta f(\boldsymbol{w}^{(i)}) = \boldsymbol{g}^{(i)}$. Метод починає роботу в точці $\boldsymbol{w}^{(0)}$ і, поки не задоволений критерій зупинки, рухається від $\boldsymbol{w}^{(i)}$ до $\boldsymbol{w}^{(i+1)}$ в тренувальному напрямку $\boldsymbol{d}^{(i)} = -\boldsymbol{g}^{(i)}$. Таким чином, ітерації градієнтного методу відбуваються наступним чином:

$$w^{(i+1)} = w^{(i)} - \boldsymbol{g}^{(i)}\eta^{(i)}, \text{ для } i = 0,1, \ldots, \qquad (1.5)$$

де η є тренувальною швидкістю. Це значення може бути фіксованим або знайдено шляхом одновимірної оптимізації вздовж напрямку навчання на кожному кроці. Оптимальне значення для швидкості навчання, одержуваної шляхом мінімізації лінії на кожному наступному кроці, зазвичай найбільш прийнятний. Тим не менш, є ще багато програмних інструментів, які використовують тільки фіксоване значення для швидкості навчання.

Алгоритм навчання градієнтним спуском має серйозний недолік – він вимагає багато ітерацій для функцій, що мають довгі вузькі структури долин. Дійсно, градієнтний спуск – це напрямок, в якому функція втрат зменшується найшвидше, але це не обов'язково призводить до найшвидшої збіжності. [10]

1.5.2 Генетичні алгоритми

НМ мають величезну проблему – гіперпараметри.

Гіперпараметри – це змінні, які визначають структуру мережі (наприклад, кількість прихованих шарів) і змінні, які визначають, як мережа навчається (наприклад, коефіцієнт швидкості навчання). Гіперпараметри встановлюються перед навчанням.

Приклади деяких гіперпараметрів: відсоток дропаутів, ініціалізація ваг мережі, функції активації, коефіцієнт швидкості навчання, імпульс, кількість епох, розмір батчу.

Використання градієнтних методів не дозволяє динамічно навчити або знайти ці параметри. Однак можливе використання генетичних алгоритмів для знаходження кращих значень гіперпараметрів. Також ми можемо навчити і самі ваги НМ.

Щоб вирішити проблему гіперпараметрів, робиться наступне [13]:

– Створюється популяція з декількох НМ.
– Призначаються випадкові (в діапазоні) гіперпараметри для всіх НМ.
– Виконуються наступні кроки для деякої кількості ітерацій.

1. Тренуються всі НМ одночасно або по одному.
2. Після того, як всі вони пройдуть навчання, розраховується їх вартість навчання.
3. Розраховується фітнес-функція (пристосованість) кожної НМ на основі її вартості. Це значення буде використовуватися, щоб збільшити шанси "розмноження" НМ. Чим вище пристосованість, тим вище ймовірність її розмноження.
4. Знайти максимальну придатність в популяції (потрібно для кроку 5, може бути проігноровано в залежності від реалізації кроку 5).
5. Вибирається дві НМ, заснованих на системі ймовірностей щодо їх придатності.

6. Кросовер генів цих двох НМ. Це створить «дочірню» НМ. Ця НМ повинна мати деякі властивості першої НМ, а деякі – другий. Цей процес також має багато реалізацій
7. Мутуються гени дочірньої НМ. Мутація необхідна для підтримки певної міри випадковості в генетичному алгоритмі.
8. Виконуються кроки 5-7 для кількості нейронних мереж в популяції. Створені «діти» зберігаються в новій популяції і призначається нова популяція змінної, що містить стару популяцію.

Виконавши всі описані вище кроки, в кінці останнього покоління алгоритм знайде популяцію, яка містить НМ з оптимальними гіперпараметрами. Це буде сама пристосована НМ з усіх в популяції [11].

Це гарне рішення для навчання гіперпараметрів, але воно має свої недоліки. Дві найбільш значимі – проблеми обчислювальних ресурсів і часу.

Навчання декількох НМ одночасно або по одному кілька разів вимагає багато часу і обчислювальних ресурсів. Це обмежує реалізацію цього рішення тільки тими людьми, які готові вкладати гроші і купувати багато обчислювальної потужності.

Це також причина, чому він широко використовується великими компаніями.

ВИСНОВКИ ДО РОЗДІЛУ

Нейроні мережі – це сучасне рішення складних проблем обробки та аналізу даних. Вони використовуються в багатьох сферах нашого життя і цей список постійно поповнюється. Завдяки цьому було винайдено багато архітектур НМ для вирішення різного класу задач. Багато з сучасних архітектур та алгоритмів були винайдені нещодавно, хоча існують і використовуються перероблені алгоритми минулого століття. Незважаючи на велике різноманіття мереж, моделей вони досі мають невирішені проблеми: підбір гарної топології, налаштування гіперпараметрів, вибір алгоритму навчання.

Для кожної з перерахованих проблем постійно знаходять часткові рішення або методи обходу, також сфера створення потужного технічного обладнання розвивається досить швидко, тому тренування моделей стає більш швидким.

Але в цілому дана сфера не є повністю налаштованою і завершеною, зараз постійно проходить її розвиток, еволюція, змінення, що тільки підтверджує актуальність роботи.

РОЗДІЛ 2 ГІБРИДНІ НЕЙРОННІ МЕРЕЖІ. МЕТОДИ СТРУКТУРНО-ПАРАМЕТРИЧНОГО СИНТЕЗУ

2.1 Гібридні нейронні мережі. Основні поняття

Побудова комбінованих нейронних мереж, які складаються з різних типів НМ, кожна з яких навчається за певним алгоритмом в багатьох випадках дозволяє значно підвищити ефективність функціонування НМ. Дослідження принципів гібридизації НМ, нечіткої логіки та генетичних алгоритмів дозволяє створювати нові топології НМ, які мають більш високу якість розпізнавання у разі одночасного зниження обчислювальних витрат на навчання.

Гібридною системою є система, яка має в собі дві чи більше інтегровані різнорідні підсистеми, які мають загальну ціль або схожі діями (при цьому ці підсистеми можуть бути різної природи).

Здатність нейронних мереж виконувати завдання, які в іншому випадку виявилися б важко розв'язуваними або такими що важко піддаються символьним обчисленням, тепер визнано, і вони часто використовуються як модулі в інтелектуальних гібридних системах.

Роботи останніх років засвідчили, що використання в машинному навчанні методів, що відповідають одній науковій парадигмі при вирішенні складних завдань і проблем, не завжди призводить до успіху. У гібридній архітектурі, що поєднує кілька парадигм або структур, ефективність одного підходу може компенсувати слабкість іншого [14]. Таким чином, при комбінуванні різних підходів, можливо обійти недоліки одної структури завдяки перевагам іншої [15].

Тому однією з головних тенденцій сучасного простору машинного навчання і програмування стала розробка гібридних систем. Таким системам притаманне поєднання різних елементів (компонентів) для досягнення однієї цілі. Використання гібридних методів і технологій дозволяє вирішувати складні проблеми, які неможливо або дуже тяжко вирішити будь-якими окремими

методами чи технологіями. Гібридизація, як фундаментальна особливість складної системи, передбачає не тільки інтеграцію, а й взаємну адаптацію та спільний розвиток її компонентів, створюючи тим самим нові особливості, не особливо специфічні для її компонентів.

В даний час здатність нейронних мереж виконувати завдання, які інакше було б важко вирішити або важко описати класичними методами, швидко росте, тому вони часто використовуються як складові компоненти для інтелектуальних гібридних систем.

2.2 Методи побудови гібридних нейронних мереж

2.2.1 Гібридні нейро-нечіткі мережі

Харківські науковці під керівництвом Є. В. Бодянського створюють гібридні нейро-нечітких-мережі, які об’єднують в собі універсальні апроксимуючі властивості традиційних НМ та легке людське розуміння систем нечіткого висновку. Подальшим розвитком нейро-нечітких-мереж стала їх гібридизація з методами теорії вейвлет-перетворення. Такі системи отримали назву вейвлет-нейро-нечітких систем. Враховуючи можливості гібридних вейвлет-нейро-нечітких систем обчислювального інтелекту, щодо їх апроксимуючих властивостей, людської зрозумілості, можливості виявляти локальні особливості оброблюваних даних і здатності реалізації м’яких обчислень на основі нечітких-множин в роботах [16, 17] було реалізовано створення гібридних адаптивних еволюційних вейвлет-нейро-нечітких систем.

Роботи [18, 19] ввели архітектуру нового гібридного W-нейрона (вейвлона). Останній використовує багатовимірні адаптивні вейвлет-функціх активації-належності. Для навчання вейвлону бели розроблені модифіковані квазі-н’ютоновські методи, що мають фільтрувальні та згладжувальні властивості. Також у роботах був створений алгоритм навчання з міцними критеріями, що дозволяє досліджувати нестаціонарні сигнали з аномальними викидами з

ненормального розподілу. Створена структура вейвлона може бути застосована як окрема гібридна мережа або як елемент вейвлет-нейро-нечітких систем. В цілому це дозволяє покращити властивості апроксимації, передбачення завдяки багатовимірним вейвлет-функціям та навчанню вагів послідовно.

2.2.2 Модулі нейронних мереж

Однак створення і навчання однієї великої глибокої НМ має безліч пов'язаних проблем: перенавчання, налаштування і оптимізація гіперпараметрів, потрібне високопродуктивне обладнання і т. д. Тому іншою гілкою розвитку НМ і глибокого навчання стало використання модульних топологій: одна велика глибока мережа, яка складається з декількох невеликих базових мереж, кожна з яких може виконувати окрему функцію (наприклад, кластеризація, зменшення розмірності даних, уточнення результатів, як і, власне, саме рішення поставленого завдання). Ця топологія є більш керованою, зрозумілою та гнучкою, порівняно з однією об'ємною мережею.

Топологія модуля включає послідовне поєднання декількох різних архітектур нейронної мережі (рис. 2.1). Загалом, модуль працює так само, як і окрема НМ. Його перевагою є поєднання різних перетворень даних, що дозволяє отримати більш точні результати.

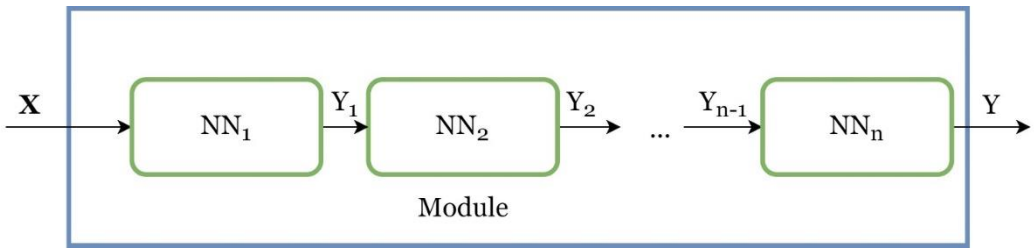


Рис. 2.1 Топологія модуля

Першими моделями модуля мають бути структури, які роблять швидкі перетворення вхідних даних в більш продуктивний та компактний вигляд, щоб подальші крупні мережі були меншими, швидшими і точнішими. Можливі моделі для використання на початку модуля:

– мережа Кохонена [20];
– (варіаційний) автоенкодер [21];
– метод *k*-сусідів [22].

Кожна з цих моделей зменшить розмір даних, а 1 і 3 варіант ідеально підходять для задач класифікації, оскільки вони роблять кластеризацію даних.

Для вихідних моделей модулю підходять невеликі НМ для уточнення результатів. Прикладом може бути використання МГУА на виході модуля для задач апроксимації. Для задач класифікації можливо використання невеликого дерева прийняття рішень.

Приклад реальної архітектури модуля наведений на зображенні (рис. 2.2).

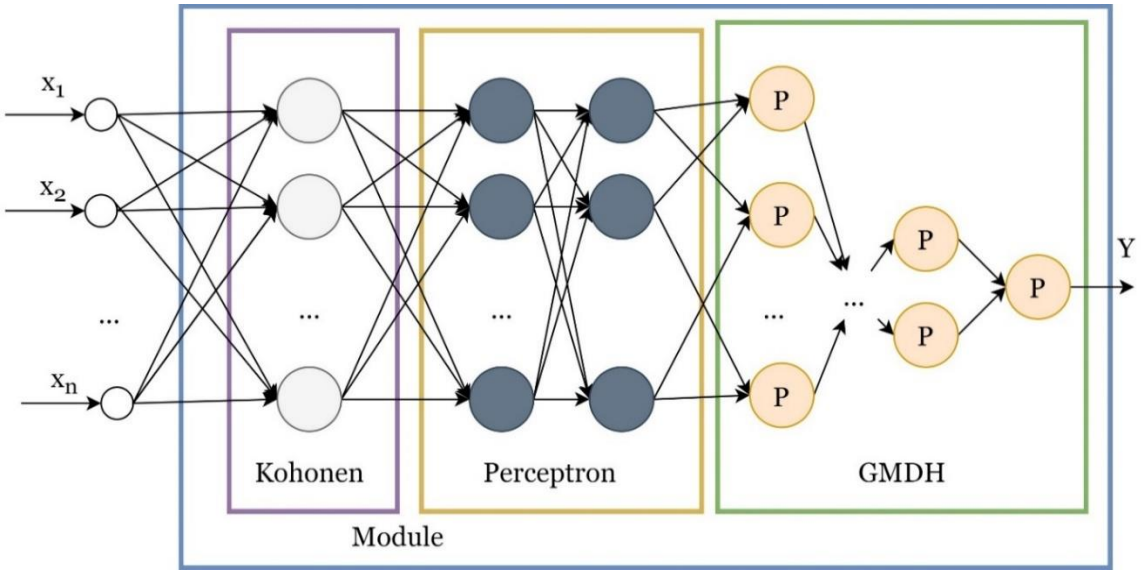


Рис. 2.2 Приклад реальної архітектури модуля

2.2.3 Гібридні нейрони LSTM та GRU

Іншою гілкою створення гібридних НМ є модифікація штучного нейрону. Одними з найпопулярніших модифікацій зараз є LSTM [23] та GRU [24] нейрони, використовуючи які створюють рекурентні нейронні мережі. Рекурентні нейронні мережі страждають від короткочасної пам'яті. Якщо послідовність достатньо довга, їм буде важко переносити інформацію від попередніх часових кроків до пізніших. Тож якщо ви намагаєтеся обробити абзац тексту, щоб робити

прогнози, RNN може втрачати початкову важливу інформацію. Архітектури нейронів LSTM та GRU вирішують цю проблему.

Стандартна архітектура LSTM-нейрону показана на зображенні (рис. 2.3). У LSTM є три основних види вузлів, які називаються гейтами: вхідний (input gate), забуваючий (forget gate) і вихідний (output gate), а також власне рекурентна комірка з прихованим станом. Крім того, в LSTM часто додають ще так звані замкові щілини (peepholes) – додаткові з'єднання, які збільшують зв'язність моделі.

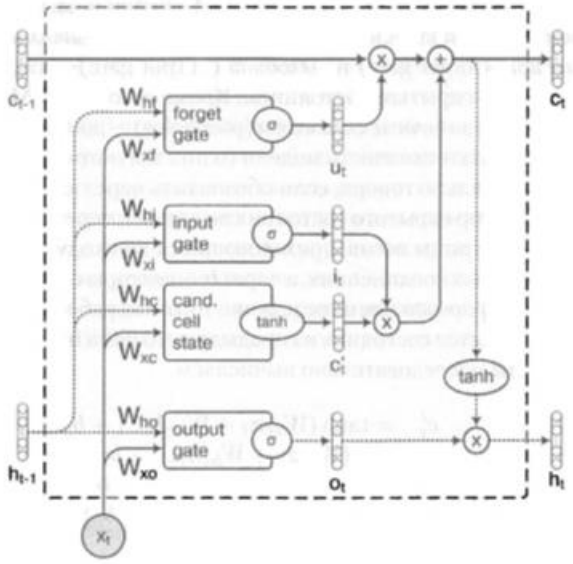


Рис. 2.3 Архітектура LSTM-нейрону [9]

LSTM має можливість видаляти або додавати інформацію до стану комірок, ретельно регулюючись гейтами.

Гейти – це спосіб опціонально відкидати інформацію. Вони складаються з сигмоподібного нейронного сіткового шару та операції по множенню вгору.

Сигмоїдний шар виводить числа між нулем і одиницею, описуючи, скільки кожного компонента слід пропустити. Значення нуля означає "не пропускайте нічого", тоді як значення одного означає "пропустити все!".

Але архітектура LSTM вимагає досить значних ресурсів. Тому була створена її модифікація у вигляді GRU-нейронів (рис. 2.4). У цій архітектурі

використовується ідея поєднання вихідного і забуває гейта, а прихований стан поєднаний зі значенням пам'яті.

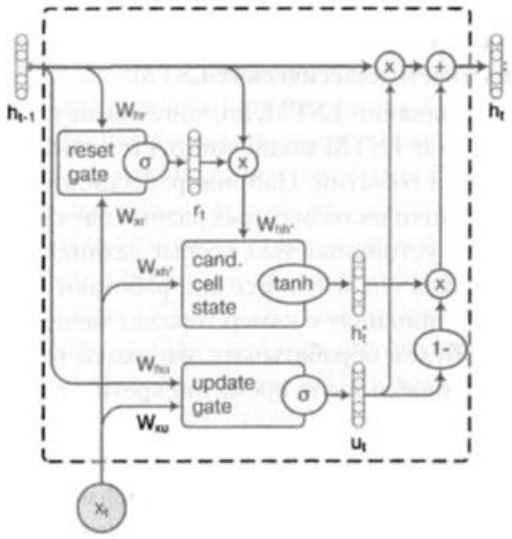


Рис. 2.4 Архітектура GRU-нейрону [9]

Основна різниця між GRU і LSTM полягає в тому, що GRU намагається зробити двома гейтами те ж саме, що LSTM робить трьома.

Практика показує, що GRU практично завжди працює так само або майже так само добре, як LSTM. А параметрів у нього виходить набагато менше: шість матриць ваг проти восьми у найпростішому варіанті або одинадцяти в більш часто використовується (з замковими щілинами). Через значно меншого числа параметрів тренувати GRU простіше, ніж LSTM, тому в деяких задачах GRU показує результати краще. Більш того, можна дозволити собі вмістити в тій же пам'яті і з тими ж ресурсами мережі з більшого числа GRU-нейронів, ніж LSTM. Тому GRU часто використовуються замість класичних LSTM. Тим не менш, кожен з цих гібридних нейронів приніс багато нових революційних результатів в простів машинного навчання: обробка послідовностей даних, даних, які залежать від часу, більш точна обробка тексту, музики і т. п.

2.2.4 Encoder-decoder [26]

Архітектура такої гібридної моделі поєднує в собі модульну топологію з гібридними рекурентними нейронами. Топологія приведена на зображенні (рис. 2.5).

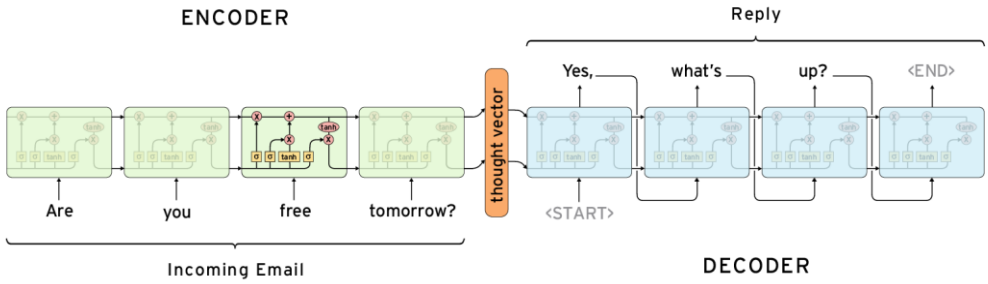


Рис. 2.5 Топологія моделі encoder-decoder

Ця модель часто використовується для обробки природньої мови. Така модель стискає весь натренований і запам’ятований стан в один вектор, що є виходом першої моделі encoder. Далі з цього вектору вже формується потрібний вихід другою моделлю в модулі – decoder’ом.

Одним з основних використань таких моделей в обробці природних мов стала задача машинного перекладу. Ми ж не переводимо слово за словом, а скоріше будуємо для себе якесь «семантичне уявлення», яке ми потім «розгортаємо» на іншій мові. Ця інтуїція в точності відповідає encoder-decoder архітектурі.

Але у encoder-decoder архітектури з є важливий недолік: всі речення цілком доводиться «згортати» в вектор фіксованої розмірності. Це означає, що довге речення буде набагато складніше згорнути і розгорнути, ніж короткий. І дійсно, практика показує, що зі збільшенням довжини входу якість падає радикально; сильно збільшувати розмір прихованого уявлення теж не виходить, тому що тоді модель стає занадто великою, і її не виходить навчати. Але все ж таки модифікована модель використовується зараз і є доволі популярною.

Можливе відокремлення частини encoder від повної архітектури і використання цих стиснутих уявлень вхідних даних для роботи з іншими архітектурами НМ і побудови іншого модуля.

2.2.5 Ансамблі нейронних мереж

Метою ансамблевих методів є поєднання різних моделей у мета-модель, яка має кращі показники узагальнення, ніж кожен окрема модель.

Така зв'язка дозволяє компенсувати недоліки однієї архітектури перевагами іншого, що є неможливим при використанні тільки однієї мережі і є безумовною перевагою. Компонентом ансамблю може бути і модуль, таким чином буде поєднана паралельна і послідовна архітектури в одну гнучку модель.

Основні причини помилок у моделях навчання обумовлені шумом, зміщенням та дисперсією.

Ансамблеві методи допомагають мінімізувати ці фактори. Ці методи розроблені для підвищення стабільності та точності алгоритмів машинного навчання.

Ансамблева топологія ГНМ (рис. 2.6) дозволяє об'єднати зусилля декількох незалежних моделей для вирішення поставленого завдання.

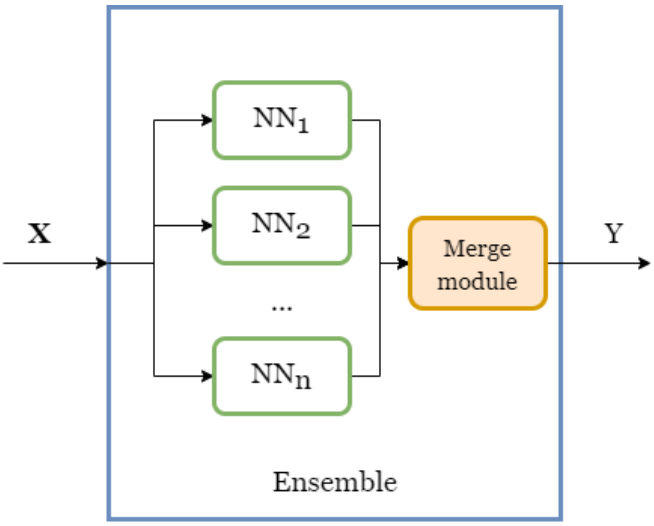


Рис. 2.6 Загальна ансамблева топологія

Об'єднання декількох ГНМ дозволяє обробити дані з різних боків, за різними особливостями. При даному виді синтезу не має значення алгоритм навчання, складність або архітектура кожної мережі, вони повністю незалежні. Кожна мережа вирішує поставлене завдання по-своєму і надає кінцеві рішення. Далі модуль об'єднання дозволяє зібрати і проаналізувати результати, отримані від кожної мережі, і з використанням алгоритмів виробити остаточне рішення поставленого завдання.

Якщо підняти рівень абстракції для ансамблевої топології, то ми отримаємо таку ж одиницю гібридної системи, як і модуль або окрема НМ. Ми подаємо на вхід певні дані і отримуємо на вихід результат.

Завдяки використанню абстракцій в програмному забезпеченні (за типом класів) є легка можливість управління і композиції представлених вище топологій.

2.3 Постановка задачі структурно-параметричного синтезу ансамблів

У разі, якщо модифікація топології штучної нейронної мережі, обрана для вирішення завдання, не дає відчутних результатів, необхідний синтез нової топології. У даній роботі розглядається ансамблева структура об'єднання мереж.

Поставимо задачу синтезу гібридної нейронної мережі на основі ансамблевої топології наступним чином.

Задано скінченний набір $D = \{(X_j, Y_j)\}, j = 1,..., P$ пар типу «атрибут-значення», де X_j, Y_j вхідний і вихідний вектор НМ, відповідно. Також задано множину гібридних нейронних мереж $N = \{n_1,..., n_m\}$.

Необхідно синтезувати таку оптимальну структуру ансамблю ГНМ на основі навчальної вибірки *D*, яка забезпечувала б ефективне розв’язання прикладної задачі (класифікації, апроксимації, прогнозування). Векторний критерій оптимальності визначається як

$$\boldsymbol{I} = \{I_1(N,D), I_2(N,D), I_3(N)\} \to \min, \qquad (2.1)$$

де $I_1(N,D) = E_{узаг}(N,D)$ похибка узагальнення, що визначає величину похибки розв'язання поставленої задачі на тестовій вибірці згенерованим ансамблем; $I_2(N,D) = T(N,D)$ – час навчання ансамблю ГНМ. $I_3(N) = S(N)$ – загальний розмір множини ГНМ (кількість мереж, кількість нейронів, зв'язків, шарів).

Таким чином, необхідно визначити оптимальну структуру і топологію ансамблю гібридних нейронних мереж, для якої відповідно до завдання (класифікація, апроксимація) і доступною навчальної вибіркою розробити алгоритм навчання для мінімізації часу навчання, мінімізації розміру ансамблю і мінімізації одержуваної помилки (або максимізації одержуваної точності). Для завдання класифікації процентне співвідношення між правильно класифікованими об'єктами і повним числом об'єктів в наборі даних використовується для визначення точності. Для завдання апроксимації для визначення точності використовувати MAPE.

ВИСНОВКИ ДО РОЗДІЛУ

Гібридизація інтелектуальних систем в сучасному просторі машинного навчання і програмування є популярним рішенням отримання кращих результатів вирішення складних проблем. Гібридизація в межах НМ передбачає модифікацію штучних нейронів, шарів, а структурно-параметричний синтез передбачає поєднання декількох НМ в єдину топологію (наприклад, модульну, ансамблеву).

Модифікація штучних нейронів передбачає розширення його архітектури, додавання додаткових вагів, функцій, зміна вигляду нелінійності, додавання зв'язків пам'яті.

Модульна топологія дозволяє розбити велику мережу на декілька послідовних невеликих мереж, кожна з яких виконуватиме свій тип обробки вхідної інформації і подавати на вхід наступної мережі.

Ансамблева топологія передбачає паралельне поєднання різних незалежних ГНМ в одну структуру і агрегацію результатів отриманих к кожної моделі для фінального рішення. Це дозволяє подавити недоліки однієї моделі, особливостями обробки іншої.

Задача синтезу ансамблів полягає в знайденні таких моделей та процедури їх поєднання і агрегації, яка б максимізувала отримувану точність і мінімізувала витрати часу та пам’яті.

РОЗДІЛ 3 СТРУКТУРНО-ПАРАМЕТРИЧНИЙ СИНТЕЗ АНСАМБЛІВ НЕЙРОННИХ МЕРЕЖ

3.1 Основні ансамблеві архітектури

3.1.1 Bagging (Bootstrap AGGregatING) [27]

В алгоритмах ансамблю методи bagging формують клас алгоритмів, які будують декілька примірників black-box оцінювача на випадкових підмножинах оригінального навчального набору, а потім агрегують їх індивідуальні прогнози, щоб сформувати остаточний прогноз. Ці методи використовуються як спосіб зменшити дисперсію базового оцінювача (наприклад, дерева рішень), ввівши випадковість в процедуру побудови, а потім зробити ансамбль із нього. У багатьох випадках bagging методи є дуже простим способом удосконалення єдиної моделі, не вимагаючи адаптації базового алгоритму. Оскільки вони надають спосіб зменшити перенавчання, методи bagging'у найкраще працюють із сильними та складними моделями (наприклад, повністю розвиненими деревами рішень), на відміну від методів boosting, які зазвичай найкраще працюють із слабкими моделями (наприклад, неглибокими деревами прийняття рішень).

Bagging predictors – це методи генерації декількох версій прогнозувача та їх використання для отримання агрегованого прогнозувача. Використовується агрегація середнього при прогнозуванні числового результату і множинне голосування при прогнозуванні класу. Кілька версій формуються шляхом виготовлення бутстрап копій навчального набору (рис. 3.1) та використання їх як нових навчальних наборів. Тести на реальних та симульованих наборах даних з використанням дерев класифікації та регресії та вибору підмножин у лінійній регресії показують, що bagging може принести значні переваги в точності. Важливим елементом є нестабільність методу прогнозування. Якщо збурення

навчального набору може спричинити значні зміни в побудованому прогнозувачі, то bagging може підвищити точність.

Sample indices	Bagging round 1	Bagging round 2	…
1	2	7	…
2	2	3	…
3	1	2	…
4	3	1	…
5	7	1	…
6	2	7	…
7	4	7	…
	↓	↓	↓
	C_1	C_2	C_m

Рис. 3.1 Bootstrap вибірки для Bagging ансамблю [8]

3.1.2 Boosting

При використанні бустингу ансамбль складається з дуже простих базових класифікаторів, які також часто називають слабкими, які часто мають лише невелику перевагу в порівнянні з випадковими здогадами. Типовим прикладом слабкого класифікатору є пень з дерева рішень. Основна концепція стимулювання полягає в тому, щоб зосередити увагу на прикладах навчання, які важко класифікувати, тобто дозволити слабким моделям згодом вчитися на неправильно класифікованих прикладах навчання для покращення ефективності роботи ансамблю. Далі буде представлено алгоритмічну процедуру, що стоїть за загальною концепцією boosting та прикладна система AdaBoost [28].

Основним принципом AdaBoost є встановлення послідовності слабких моделей на неодноразово змінених версіях даних. Прогнози всіх з них потім поєднуються за допомогою зваженої більшості голосів (або суми) для отримання

остаточного прогнозу. Модифікації даних при кожній так званій boosting ітерації складаються із застосування ваг w_1, w_2, …, w_N до кожного з навчальних зразків. Спочатку ці ваги налаштовані $w_i=1/N$, так що перший крок просто навчає слабкого учня за початковими даними. Для кожної наступної ітерації вибіркові ваги індивідуально змінюються, а алгоритм навчання повторно застосовується до повторно зважених даних. На даному кроці ті приклади тренувань, які були неправильно прогнозовані boosting моделлю, індукованою на попередньому кроці, збільшують їх вагу, тоді як ваги зменшуються для тих, які були прогнозовані правильно. У процесі ітерацій приклади, які важко передбачити, отримують все більший вплив. Кожен наступний слабкий класифікатор змушений концентруватися на прикладах, пропущених попередніми в послідовності (рис. 3.2).

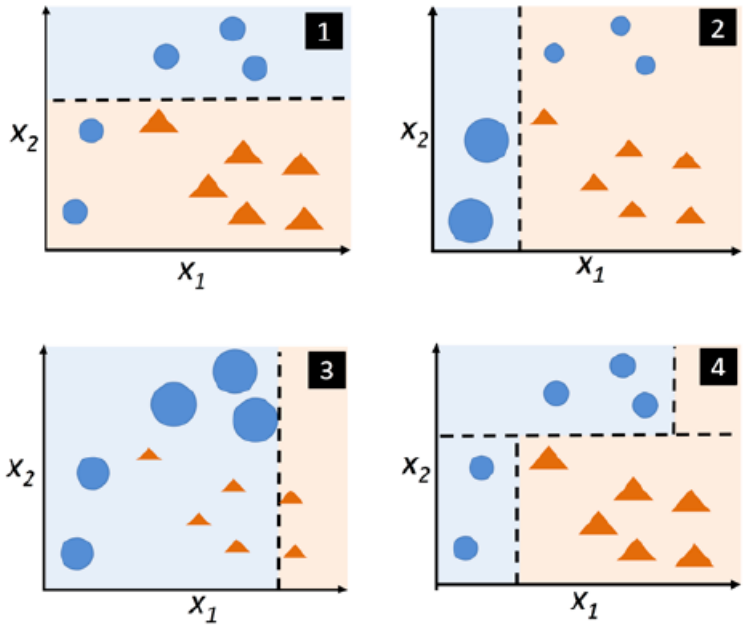


Рис. 3.2 Приклад роботи AdaBoost [8]

3.1.3 Stacking [29]

Stacked Generalization or “Stacking” – це алгоритм машинного навчання ансамблю, який включає комбінування прогнозів з декількох моделей машинного навчання на одному і тому ж наборі даних, таких як bagging та boosting.

Stacking вирішує питання:

– З огляду на кілька моделей машинного навчання, які вміло вирішують проблему, але різними способами, як ви обираєте, яку модель використовувати (довіряти)?

Вирішення цього питання полягає у використанні іншої моделі машинного навчання, яка навчається, коли використовувати та довіряти кожній моделі в ансамблі.

– На відміну від bagging'у, при стекінгу моделі, як правило, різні (наприклад, не всі дерева рішень) і навчаються на одному наборі даних (наприклад, замість вибірок з навчального набору даних).
– На відміну від бустингу, при стекінгу одна модель використовується, щоб навчитися найкраще поєднувати прогнози з моделей, що надають внесок (наприклад, замість послідовності моделей, що виправляють прогнози попередніх моделей).

Архітектура моделі укладання включає дві або більше базових моделей, які часто називають моделями рівня 0, і мета-модель, що поєднує в собі прогнози базових моделей, іменується як модель рівня 1 (рис. 3.3).

– Моделі рівня 0 (базові моделі): моделі підходять до даних тренувань і прогнози яких складаються.
– Модель 1-го рівня (мета-модель): модель, яка навчається як найкраще поєднувати прогнози базових моделей.

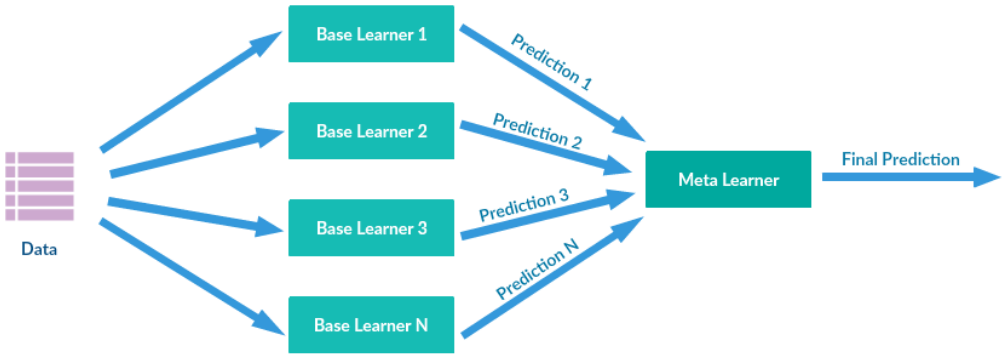


Рис. 3.3 Архітектура Stacking ансамблю [30]

Мета-модель навчається на прогнозах, зроблених базовими моделями на тестових даних. Тобто, дані, які не використовуються для підготовки базових моделей, подаються до базових моделей, робляться прогнози, і ці прогнози, поряд із очікуваними результатами, забезпечують вхідні та вихідні пари навчального набору даних, використовуваних для підгонки мета-моделі.

Виходи з базових моделей, що використовуються як вхід до мета-моделі, можуть бути реальними значеннями у випадку регресії, а також значення ймовірності, імовірнісні значення або позначки класу у разі класифікації.

Дані тренінгу для мета-моделі можуть також включати входи до базових моделей, наприклад вхідні елементи навчальних даних. Це може надати додатковий контекст мета-моделі щодо того, як найкраще поєднувати прогнози з мета-моделі.

Після того, як навчальний набір даних підготовлений для мета-моделі, мета-модель може бути навчена ізольовано на цьому наборі даних, а базові моделі можуть бути навчені на всьому початковому наборі даних тренінгу.

Стекінг доцільний, коли кілька різних моделей машинного навчання описують різні особливості набору даних. Інший спосіб сказати це, що передбачення, зроблені моделями, або помилки в прогнозах, зроблені моделями, є некорельованими або мають низьку кореляцію.

Базові моделі часто складні та різноманітні. Таким чином, часто корисно використовувати цілий ряд моделей, які роблять дуже різні припущення щодо вирішення завдання передбачуваного моделювання, такі як лінійні моделі, дерева рішень, векторні машини підтримки, нейронні мережі тощо. Інші алгоритми ансамблю можуть також використовуватися в якості базових моделей, таких як random forests [31].

В якості базових моделей використовуються різноманітний діапазон моделей, які роблять різні припущення щодо завдання передбачення.

Мета-модель часто проста, забезпечує чітку інтерпретацію прогнозів, зроблених базовими моделями. Таким чином, як мета-модель часто використовують лінійні моделі, такі як лінійна регресія для регресійних завдань (прогнозування числового значення) та логістична регресія для завдань класифікації (прогнозування мітки класу). Хоча цей підхід є популярним, але він не обов'язковий.

– Мета-модель регресії: лінійна регресія.
– Мета-модель класифікації: логістична регресія.

Стекінг призначений для поліпшення продуктивності моделювання, хоча не гарантується, що це призведе до покращення у всіх випадках.

Досягнення підвищення ефективності залежить від складності проблеми та того, чи вона достатньо добре представлена навчальними даними та достатньо складна, що є чому навчитися, комбінуючи прогнози. Це також залежить від вибору базових моделей та того, чи є вони достатньо вмілими та достатньо некорельованими у своїх прогнозах (або помилках).

Якщо базова модель виконує так само, або краще, ніж ансамбль, замість нього слід використовувати базову модель, враховуючи її меншу складність (наприклад, її простіше описати, навчати та підтримувати).

3.1.4 Random forests [31]

Random forests – це така комбінація дерев прийняття рішень, що кожне дерево описує випадкову підмножину ознак навчальної вибірки, ця підмножина залежить від значень випадкового вектору, відібраного незалежно та з однаковим розподілом для всіх дерев у лісі. Похибка узагальнення для лісів сходиться до можливого ліміту, коли кількість дерев у лісі стає великою. Похибка узагальнення лісу класифікаторів дерев залежить від потужності окремих дерев у лісі та співвідношення між ними. Внутрішні оцінки відстежують помилки, міцність та кореляцію, і вони використовуються для показу реакції на збільшення кількості функцій, що використовуються при розділенні. Внутрішні оцінки також використовуються для вимірювання змінної важливості. Ці ідеї також застосовні до регресії.

Випадкові ліси є ефективним інструментом прогнозування. Через Закон про великі числа вони не перенавчаються. Введення правильного виду випадковості робить їх точними класифікаторами та регресорами. Використання тестової вибірки конкретизує теоретичні значення сили та кореляції. Приклад такого лісу наведений на зображенні (рис. 3.4).

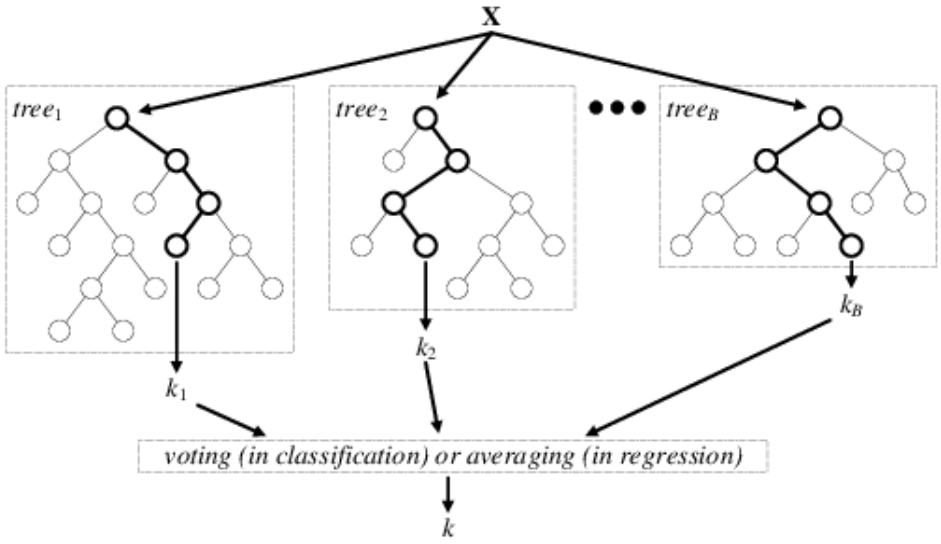


Рис. 3.4 Приклад Random Forest Classifier [32]

3.1.5 Gradient Boosting [33]

Gradient Boosting – це технологія машинного навчання проблем регресії та класифікації, яка виробляє модель передбачення у вигляді ансамблю слабких моделей прогнозування, як правило, дерев рішень. Він будує модель поетапно, як це роблять інші прискорювальні методи, і узагальнює їх, дозволяючи оптимізувати довільну диференційовану функцію втрат.

3.2 Ансамблева топологія на основі оцінки індивідуального вкладу

Як уже згадувалося, ансамбль, як правило, має паралельну структуру із зав'язуючим шаром.

Необхідною і достатньою умовою побудови набору мереж, більш точних у вирішенні класифікаційної (прогнозуючої) задачі, ніж будь-яка окрема мережа, є включення елементів, що відповідають критеріям точності та різноманітності. Оскільки різноманітність ансамблю зменшується зі збільшенням точності членів, вирішення проблеми створення ефективного поєднання зводиться до пошуку компромісу. У багатьох методах різноманітність та точність досягаються шляхом маніпулювання даними навчальних зразків. Однією з проблем цих підходів є те, що вони, як правило, створюють великі ансамблі. Для зберігання навчених модулів (класифікаторів) потрібен великий об'єм пам'яті. Для подолання цієї проблеми використовується процедура обрізки (Pruning), яка забезпечує оптимальний вибір підмножини окремих модулів (класифікаторів) з уже побудованого набору з точки зору точності, різноманітності та споживання пам’яті.

Нехай $D = \{d_1, ..., d_N\}$ множина з N зразків даних, де $d_i = \{(\boldsymbol{x}_i, y_i) \mid i \in [1, N]\}$ пара вхідних ознак та мітка, яка представляє i-у точку даних, $C = \{c_1, ..., c_m\}$ множина класифікаторів, де $c_i(\boldsymbol{x}_j)$ робить передбачення i-го класифікатора в j-му зразку, $V = \{v^{(1)}, ..., v^{(N)} \mid v^{(i)} = [v_1^{(i)}, ..., v_L^{(i)}], i \in [1, N]\}$ множина векторів, де $v_j^{(i)}$ – кількість

передбачень для *j*-ї мітки *i*-ї точки даних ансамблю с мажоритарним голосуванням, а L – кількість вихідних міток.

Необхідно на основі точності і різноманітності класифікаторів $C = \{c_1, ..., c_m\}$ відібрати компоненти для формування ансамблю, маючи тестовий набір даних и маючи на увазі, що НМ попередньо навчені на бутстреп-вибірках.

Тому попереднім етапом створення ансамблю є навчання основних класифікаторів, які мають бути незалежними.

Ці НМ навчаються на незалежних бутстрап-вибірках. В результаті маємо наступний алгоритм:

Маємо набір тренувальних даних $(x_1, y_1), \ldots, (x_m, y_m)$ з мітками $y \in \{1, ..., k\}$.

Треба отримати *t* бутстреп-вибірок D_t.

Незалежно навчити *t* моделей h_t, кожного на власній вибірці D_t.

В загальному випадку прогнозування компонент ансамблю в одному прикладі даних можна розподілити на чотири підмножини, в яких:

– окремий класифікатор прогнозує вірно і знаходиться в групі меншини;
– окремий класифікатор прогнозує вірно і знаходиться в групі більшості;
– окремий класифікатор прогнозує невірно і знаходиться в групі меншини;
– окремий класифікатор прогнозує невірно і знаходиться в групі більшості.

Були розроблені два наступні правила для створення евристичної метрики оцінки індивідуальних вкладів моделей ансамблю:

– вірні прогнози вносять позитивні вклади, невірні – негативні;
– вірні прогнози, які входять в групу меншини, вносять більше позитивних вкладів, ніж вірні прогнози, які входять в групу більшості, а невірні прогнози, які входять в групу меншини, вносять менше негативних вкладів, ніж невірні прогнози, які входять в групу більшості.

Індивідуальний вклад класифікатора c_i визначається так:

$$IC_i = \sum_{j=1}^{N} IC_j^{(i)}, \qquad (3.1)$$

де $IC_i^{(j)}$ – вклад класифікатора c_i в j-ту точку даних d_j.

$IC_i^{(j)}$ залежить від підмножини прогнозування класифікатора.

Коли прогнозування класифікатора c_i в точці даних x_j дорівнює y_j, якщо $c_i(x_j)$ в групі меншини, то $IC_i^{(j)}$ визначається наступним чином

$$IC_i^{(j)} = 2v_{\max}^{(j)} - v_{c_i(x_j)}^{(j)}, \tag{3.2}$$

де $v_{\max}^{(j)}$ – кількість більшості голосів в d_j; $v_{c_i(x_j)}^{(j)}$ – кількість передбачень $c_i(x_j)$, які були зроблені раніше.

Коли передбачення класифікатора c_i в точці x_j рівне y_j і $c_i(x_j)$ в більшості (в цьому випадку $v_{c_i(x_j)}^{(j)} = v_{\max}^{(j)}$), $IC_i^{(j)}$ визначається як

$$IC_i^{(j)} = v_{\text{sec}}^{(j)}, \tag{3.3}$$

де $v_{\text{sec}}^{(j)}$ – друге за величиною кількість голосів на d_j мітці. $v_{\text{sec}}^{(j)} - v_{\max}^{(j)}$ є оцінкою «ступеня позитивного вкладу» в цьому випадку. Суть полягає в тому, що якщо модель прогнозує правильно, як і більшість інших моделей, то її вклад не є цінним, бо і без неї були б отримані вірні результати.

Коли $c_i(x_j)$ не дорівнює y_j $IC_i^{(j)}$ визначається як

$$IC_i^{(j)} = v_{correct}^{(j)} - v_{c_i(x_j)}^{(j)} - v_{\max}^{(j)}, \tag{3.4}$$

де $v_{correct}^{(j)}$ – кількість голосів за вірну мітку d_j. Подібно «ступеня позитивного вкладу», «ступінь негативного вкладу» оцінюється по формулі $v_{correct}^{(j)} - v_{c_i(x_j)}^{(j)}$.

Поєднавши формули (3.2), (3.3) і (3.4) за допомогою формули (3.1), ІВ класифікатора c_i визначається наступним чином:

$$IC_i = \sum_{j=1}^{N} (\alpha_{ij}(2v_{\max}^{(j)} - v_{c_i(x_j)}^{(j)}) + \beta_{ij} v_{\sec}^{(j)} + \theta_{ij}(v_{correct}^{(j)} - v_{c_i(x_j)}^{(j)} - v_{\max}^{(j)})), \quad (3.5)$$

де

$$\alpha_{ij} = \begin{cases} 1,\ c_i(x_j) = y_j, c_i(x_j) \textit{ в меншині}, \\ 0,\ \textit{інакше}; \end{cases}$$

$$\beta_{ij} = \begin{cases} 1,\ c_i(x_j) = y_j, c_i(x_j) \textit{ в більшості}, \\ 0,\ \textit{інакше}; \end{cases} .$$

$$\theta_{ij} = \begin{cases} 1,\ c_i(x_j) \neq y_j, \\ 0,\ \textit{інакше}. \end{cases}$$

Використовуючи формулу (3.5) формується множина моделей, які потрапляють в ансамбль. Далі обирається метод поєднання ансамблів результатів в єдине значення.

Можливими рішеннями є:

– мажоритарне голосування;
– вираховування зваженого значення. При цьому необхідно кожному класифікатору надати якусь вагу, базуючись чи на власних здогадках, чи на отриманих результатах на тренувальній вибірці (наприклад, обернене значення помилки)
– використання стекінгу, при цьому необхідно обрати мета-модель, яка буде обробляти отримані результати.

Загальний алгоритм створення ансамблевої архітектури НМ на основі ІВ має наступний вигляд:

1. Маємо тренувальний набір даних $(x_1, y_1), ..., (x_m, y_m)$ з мітками $y \in \{1, ..., k\}$.
2. Отримуємо t бутстреп-вибірок.
3. Незалежно навчаємо t класифікаторів h_t на власних вибірках D_t.
4. Отримуємо прогнози кожної моделі c_i по j-й точці даних d_j на тестовій вибірці.
5. Знаходимо ІВ кожного класифікатора c_i по формулі (3.5) в залежності від типу прогнозу

6. Формуємо перелік (c_i, IC_i) і сортуємо його в порядку зменшення ІВ.
7. Обираємо параметр *p,* який є бажаним відсотком класифікаторів *C,* на основі яких буде побудований підансамбль. Цей параметр обирається виходячи з наявних ресурсів, таких як пам’ять, час і кошти.
8. Створити підансамбль
9. Обрати метод поєднання та потрібні для нього невідомі параметри.

На основі алгоритму ми отримали оцінку вибраних нейронних мереж з їх індивідуальними перевагами. На практиці помилка класифікації в ансамблі, як правило, показує монотонне зменшення як функція від кількості елементів.

При великих розмірах ансамблів ці криві досягають постійного рівня асимптотичної помилки, що зазвичай вважається найкращим результатом, якого можна досягти.

Однією з проблем ансамблевих підходів є те, що їх використання призводить до створення непропорційно великих поєднань, що вимагає значного обсягу пам’яті для зберігання навчених класифікаторів та скорочення часу реакції на прогнози. Спрощення ансамблів або обрізання – це метод, який вирішує цю проблему шляхом вибору підмножини кожної мережі з підготовленого файлу для створення підгрупи передбачення. Ретельно обираються модулі в підансамблі, щоб переконатися, що він мінімально складний (щоб зменшити вимоги до пам’яті та час відгуку) та щоб точність прогнозування дорівнювала або перевищує вихідний ансамбль. Наступним кроком у створенні ефективного ансамблю є використання процедури обрізки.

Для вибору необхідних класифікаторів зазвичай використовуються такі методи, як спрощення зменшення помилки (англ. Reduce-Error Pruning), спрощення Каппа (англ. Kappa Pruning), мінімізація граничного відстані (англ. Margin Distance Minimization) і орієнтоване упорядкування (англ. Orientation Ordering). Однак усі ці підходи базуються на підборі мереж для точності чи різноманітності, які вже були оцінені на основі індивідуального внеску. Відомо

також, що для створення ефективного набору іноді недостатньо врахувати різноманітність та точність. Пропонується використовувати такий підхід, як спрощення за допомогою додаткового вимірювання (англ. Complementary Measure), який також враховує взаємодію класифікаторів.

Алгоритм спрощення.

1. Отримати прогнози кожного члена ансамблю після запуску на тестовому наборі даних.
2. Сформувати підансамбль $S_{u\text{-}1}$ з модулів, котрі на виході мають прогноз $c_i(x_j)$, який не дорівнює y_i (тобто прогнозують невірно).
3. Вибрати класифікатор з кращими показниками різноманітності і точності з отриманого списку в результаті попереднього алгоритму і додати його в подансамбль.
4. Отримати величину S_u, яка показує вплив класифікатора з найкращими результатами на «поганий» підансамбль:

$$S_u = \arg\max_k \sum_{(x,y)\in Z} I(y = h_k(\boldsymbol{x}) \; and \; H_{S_{u-1}}(\boldsymbol{x}) \neq y), \qquad (3.6)$$

де класифікатор з найкращими результатами належить до початкового ансамблю $h_k(\mathbf{x})$ and $H_{S_{u-1}}$; $I(g)$ – індикаторна функція ($I(true)$ = 1, $I(false)$ = 0); S_u – величина, згідно з якою відбираються члени в скорочений подансамбль.

5. Задаємо порогове значення для відбору класифікатора. Тобто, якщо помилка, зроблена подансамблем $S_{u\text{-}1}$ більше, ніж помилка S_u, а саме, різниця їх помилок перевищує попередньо визначене граничне значення, то класифікатор вважається доповненням і відбирається в скорочений подансамбль. Таким чином,

$$S_u = \arg\max_k \sum_{(x,y)\in Z} I(|\, y - H_{S_{u-1}}(\boldsymbol{x})\,| - |\, y - h_k(\boldsymbol{x})| > threshold). \qquad (3.7)$$

6. Повторюємо кроки 3-5 с кожною моделлю і початковим підансамблем $S_{u\text{-}1}$.

3.3 Стекінг на основі МГУА для задач апроксимації

В даній роботі пропонується використання МГУА мережі [34] в якості мета-моделі в ансамблевій топології стекінгу при розв’язанні задачі апроксимації.

Як було описано в п. 3.1.3 частіше всього для цього типу задач, як агрегуючу модель, використовують лінійну регресію. Насправді таке поєднання є звичайним зважуванням виходу кожної мережі, але з більш «розумними» вагами.

Для використання всіх можливостей архітектури стекінгу мета-модель має бути трішки складнішою, ніж лінійна регресія, але з мінімальною різницею в пам’яті і часі навчання.

МГУА є варіантом, який може замінити лінійну регресію. Цей метод має наступні переваги:

– обробка даних йде поліномами вищої степені, тобто це вже буде не просто зважження результатів апроксиматорів;
– завдяки варіюванню кількості вхідних змінних в нейрони МГУА можна визначити, що результати якогось апроксиматора насправді не використовуються в фінальному ансамблі і прибрати його;
– налаштування і пошук структури МГУА мережі проходить досить швидко і вона займає невелику частину пам’яті.

Ступінь поліному та кількість входів в нейрони МГУА можуть бути визначені на основі власних здогадок, емпірично, або просто перебором. Найчастіше 2 і 3 ступені поліному працюють найкраще. Кількість входів в нейрони залежить від кількості апроксиматорів.

3.4 Експериментальні результати

Для перевірки роботи різних архітектур пропонується вирішення задачі класифікації і апроксимації (регресії). Як критерій точності для задачі

класифікації використовується відсоткове співвідношення між правильно класифікованими об'єктами і повним числом об'єктів в наборі даних, для задачі апроксимації – MAPE.

3.4.1 Приклад 1. Класифікація.

Для перевірки точності класифікації окремих мереж і всього ансамблю використовуємо набір даних Wine. Кількість зразків в наборі – 178. Кожен об'єкт має 13 ознак, представлених дійсними числами більше нуля, і одну мітку класу. Кількість класів – 3 (табл. 3.1). Для тестової вибірки обрані 20% набору даних, відповідно, 80% – для навчання.

Таблиця 3.1. Приклад вибірки даних

Назва ознаки	#1	#2	#3
alcohol	14.23	11.82	12.2
malic_acid	1.71	1.47	3.03
ash	2.43	1.99	2.32
alcalinity_of_ash	15.6	20.8	19
magnesium	127	86	96
total_phenols	2.8	1.98	1.25
flavanoids	3.06	1.6	0.49
nonflavanoid_phenols	0.28	0.3	0.4
proanthocyanins	2.29	1.53	0.73
color_intensity	5.64	1.95	5.5
hue	1.04	0.95	0.66
od280/od315_of_diluted_wines	3.92	3.33	1.83
proline	1065	495	510
Клас	1	2	3

3.4.1.1 Архітектура ансамблю на основі оцінки індивідуальних вкладів

Розглянемо архітектури НМ, які були використані для вирішення поставленої задачі класифікації [35]:

1. Feed-forward neural network [36]

Топологія представлена на зображенні (рис. 3.5).

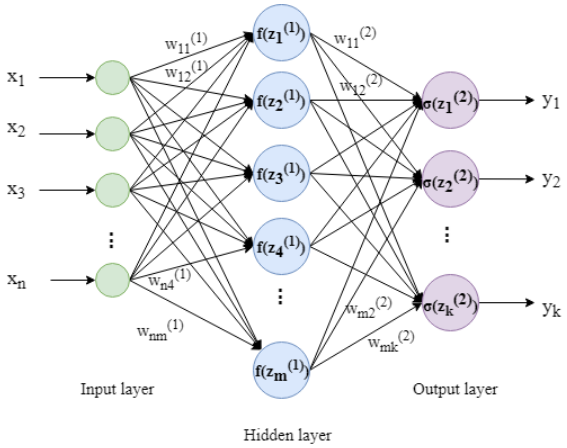


Рис. 3.5 Топологія FFNN

Кількість нейронів вхідного шару – 13, прихованого шару – 48. Функція активації прихованого шару – логістичний сигмоїд.

Як функції активації нейронів вихідного шару використовувалася функція softmax.

Оптимізаційний алгоритм – Adam.

Кількість епох навчання – 100, розмір міні-батчу – 10.

2. Радіально-базисна мережа [37]

Топологія представлена на зображенні (рис. 3.6).

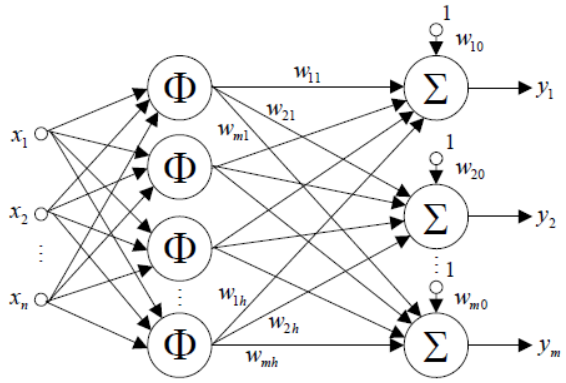


Рис. 3.6 Топологія радіально-базисної НМ

Кількість нейронів вхідного шару - 13, прихованого шару - 6.

Як ядро радіально-базисних функцій обрана функція Гаусса.

Параметри функцій Гаусса кожного нейрона прихованого шару ініціалізовані центрами 6 кластерів, знайдених з використанням алгоритму k-середніх на тренувальній вибірці.

Параметри β ініціалізовані константою 0.3.

Оптимізаційний алгоритм - Adam.

Вхідні вектора перед початком навчання були стандартизовані.

Кількість епох навчання - 100, розмір міні-батчу – 30.

3. Мережа зустрічного поширення [38]

Топологія представлена на зображенні (рис. 3.7).

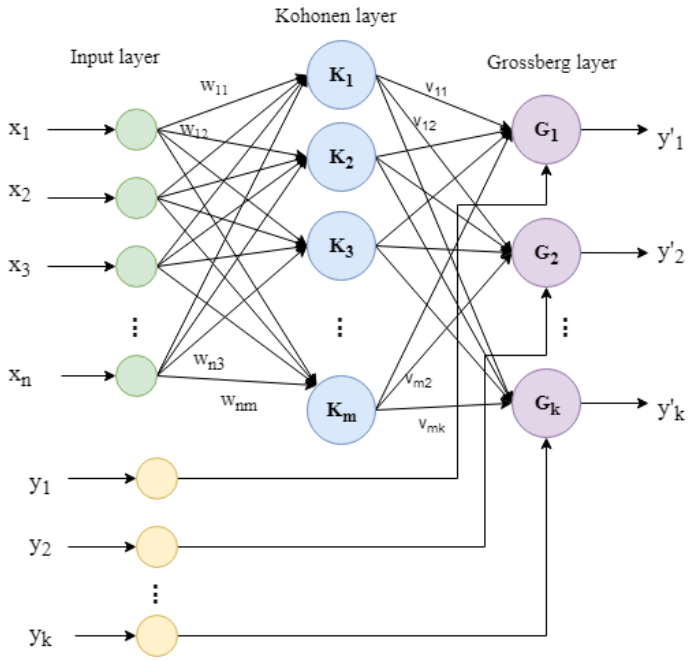


Рис. 3.7 Топологія мережі зустрічного поширення

Кількість нейронів вхідного шару – 13, прихованого шару – 3. Вхідні вектори перед початком навчання були стандартизовані і нормалізовані. Ваги шару Кохонена були ініційовані випадковими значеннями з інтервалу (0, 1) і нормалізовані.

4. Ймовірнісна нейронна мережа [39]

Топологія представлена на зображенні (рис. 3.8).

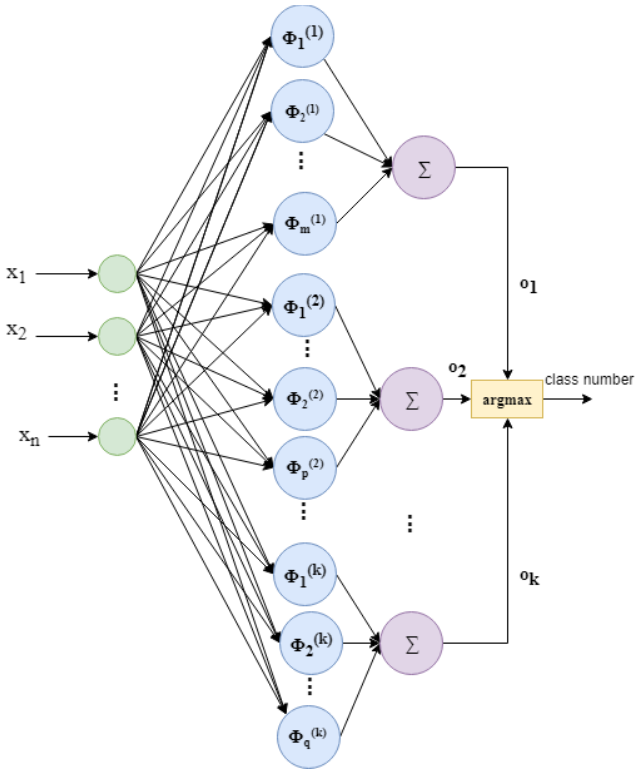


Рис. 3.8 Топологія ймовірнісної НМ

Кількість нейронів вхідного шару – 13, першого прихованого шару – 142, другого прихованого шару – 3.

5. NEFClassM [40]

Топологія представлена на зображенні (рис. 3.9).

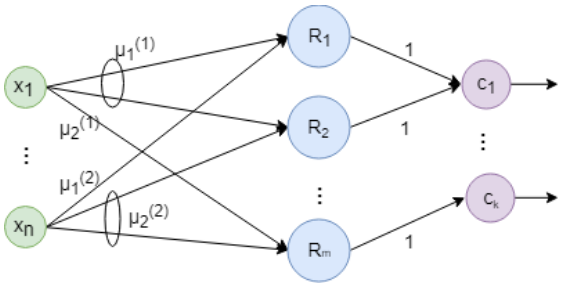


Рис. 3.9 Топологія NefClassM

Кількість вхідних нейронів – 13. Для кожної ознаки було визначено три початкових нечітких безлічі з назвами «малий», «середній», «великий», $k_{max} = 40$. В шарі правил присутні 3 нейрона. З навченої бази правил було отримано по

одному кращому правилом для кожного класу за алгоритмом. Кількість вихідних нейронів – 3.

Параметри нечітких множин навчалися градієнтним методом.

Кількість епох навчання – 100, розмір міні-батчу – 10.

6. Naïve Bayes Classifier [41]

Топологія представлена на зображенні (рис. 3.10).

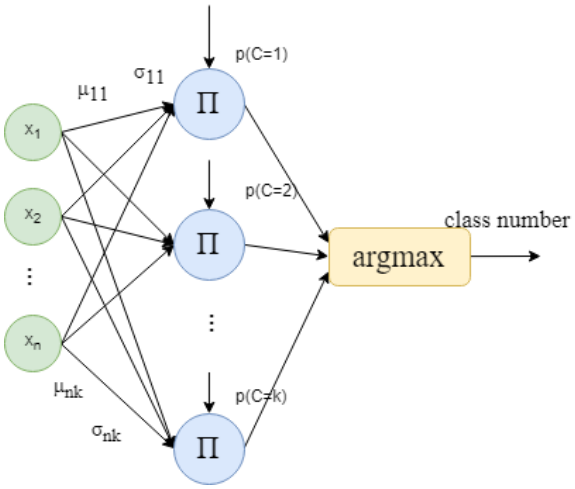


Рис. 3.10 Топологія Naïve Bayes classifier

В якості апріорного розподілу $p(C)$ була взята формула (3.8).

$$p(C=c)=\frac{N_c}{N}, \tag{3.8}$$

де N_c – кількість елементів, які мають клас c, N – загальна кількість елементів в тренувальній вибірці.

Функціями розподілу обрані нормальні розподіли.

7. Дерево прийняття рішень [42]

Топологія представлена на зображенні (рис. 3.11).

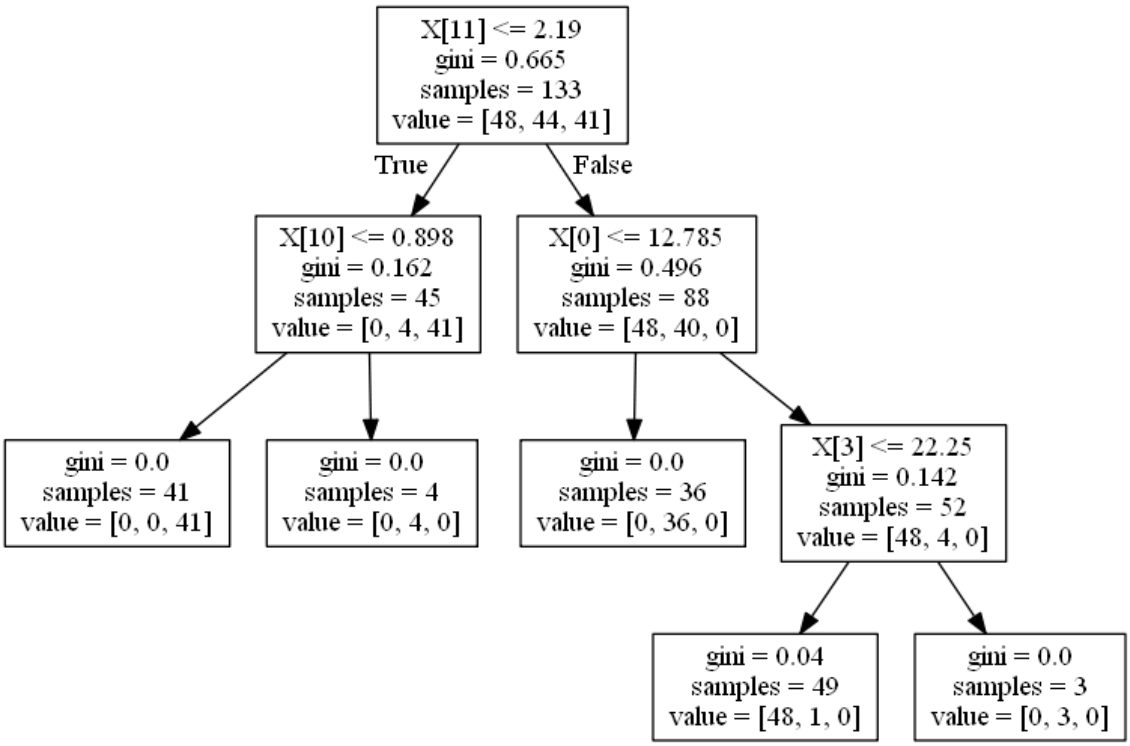


Рис. 3.11 Топологія дерева прийняття рішень

Максимальна глибина дерева – 3. Мінімальне розгалуження – 2.

Всі класифікатори були навчені на тренувальній вибірці (табл. 3.2).

Таблиця 3.2. Точність рішення окремих класифікаторів

Класифікатор	Тренувальна точність	Тестова точність
Naïve Bayes	95.49%	95.55%
RBFN	93.23%	91.11%
FFNN	90.22%	91.11%
CPN	91.73%	91.11%
PNN	90.22%	82.22%
NEFClassM	91.73%	88.88%
Decision Tree	95.49%	86.66%

Використаємо формулу (3.5) для знаходження ІВ кожної мережі на тренувальній множині (табл. 3.3).

Таблиця 3.3. Індивідуальний внесок окремих мереж

Класифікатор	ІВ
Decision Tree	50
PNN	43
NefClassM	39
Naïve Bayes	37
RBFN	34
CPN	32
Multilayer perceptron	26

Об’єднаємо все мережі в зважений ансамбль (табл. 3.4).

Таблиця 3.4. Точність рішення повного зваженого ансамбля

Тренувальна точність	Тестова точність
97.74%	95.55%

Також для порівняння використаємо звичайне мажоритарне голосування (табл. 3.5).

Таблиця 3.5. Точність повного ансамблю при мажоритарному голосуванні

Тренувальна точність	Тестова точність
99.24%	100%

Можна помітити, що об’єднання в повний ансамбль, дає набагато кращі результати, ніж окремі класифікатори, але він занадто великий, тому потрібно зменшити його розмір обранням найкращих моделей. На цьому етапі в ансамбль можуть бути відібрані моделі з найбільшими значеннями ІВ:

– naïve Bayes;
– radial-basis function network;
– NefClassM;
– probability neural network;
– decision tree.

Об’єднаємо наведені вище моделі в зважений ансамбль (табл. 3.6).

Таблиця 3.6. Точність зваженого ансамбля з найкращими мережами

Класифікатор	Тренувальна точність	Тестова точність
NB, RBFN, PNN, NefClassM, DT	99.25%	93.33%

Спостерігається деяке перенавчання при використанні мереж з найбільшими індивідуальними вкладами, тому наступним етапом стає застосування алгоритму скорочення ансамблю. Як елементи початкового ансамблю для алгоритму скорочення були обрані мережі з найгіршими показниками: FFNN і мережу зустрічного поширення.

В результаті застосування алгоритму скорочення ансамблю, отримана архітектура кінцевого ансамблю, що включає наступні класифікатори:

– naïve Bayes;
– radial-basis function network;
– NefClassM;
– decision tree.

Об’єднаємо наведені вище моделі в скорочений зважений ансамбль (табл. 3.7).

Таблиця 3.7. Точність скороченого зваженого ансамблю

Класифікатор	Тренувальна точність	Тестова точність
NB, RBFN, NefClassM, DT	98.49%	97.77%

Для порівняння об’єднаємо також в мажоритарний ансамбль (табл. 3.8).

Таблиця 3.8. Точність скороченого мажоритарного ансамблю

Класифікатор	Тренувальна точність	Тестова точність
NB, RBFN, NefClassM, DT	97.74%	95.55%

Як можна помітити, після застосування операції скорочення ансамблю не спостерігається проблема перенавчання, а точність класифікації перевершує точність будь-якого окремого класифікатора, при цьому розмір ансамблю був

скорочений майже вдвічі – до 4 моделей з початкових 7. Використання ваг в даному ансамблі дозволило значно збільшити точність класифікації, в порівнянні зі звичайним мажоритарних голосуванням.

Нижче наведена символьна топологія фінального скорченого ансамблю (рис. 3.12).

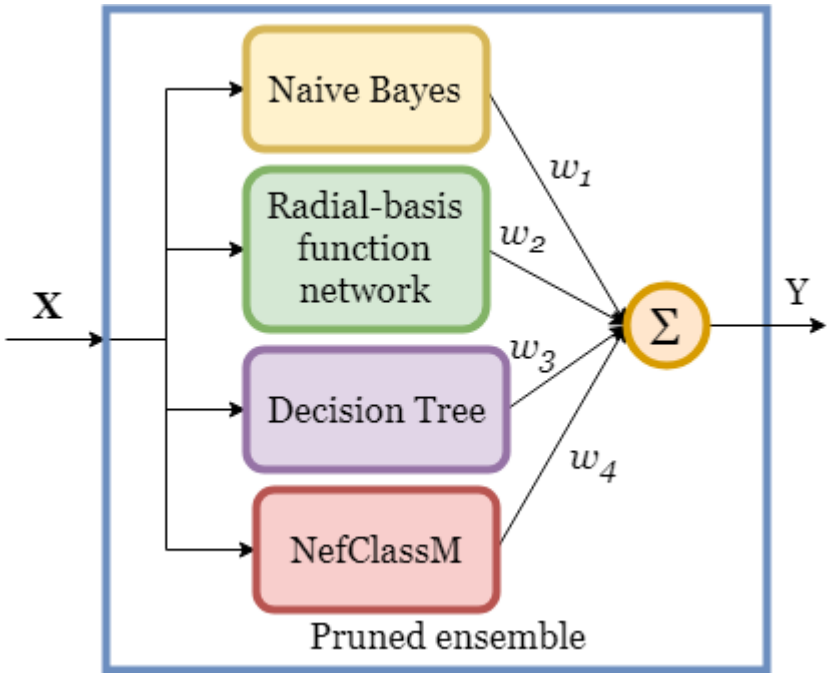


Рис. 3.12 Топологія скороченого ансамблю

3.4.1.2 Інші архітектури ансамблів

Для порівняння результатів були використані чотири інші ансамблеві структури.

– Random Forest Classifier [42]

Ансамбль містить 5 дерев-класифікаторів без обмеження на глибину кожного дерева.

– Bagging Classifier з Support Vector Classifier [43]

Ансамбль містить 5 моделей SVC, кожен має радіально-базисне ядро, параметр гамма обернено пропорційна від дисперсії тренувальної вибірки.

– AdaBoost [28]

Ансамбль містить 5 моделей. Кожна модель являє собою «слабке» дерево прийняття рішень. Під «слабким» мається на увазі те, що максимальна глибина дерева дорівнює 1.

– Gradient Boosting Classifier [33]

Ансамбль містить 5 моделей. Кожна модель є деревом прийняття рішень. Кожен ансамбль був навчений на тренувальній вибірці (табл. 3.9).

Таблиця 3.9. Результати навчання сторонніх ансамблевих структур

Ансамбль	Тренувальна точність	Тестова точність
Random Forest	99.25%	88.88%
Bagging SVC	72.1%	60%
AdaBoost	93.23%	88.88%
Gradient Boosting	97.77%	91.11%

Як можна побачити, інші ансамблеві моделі з більшою складністю класифікаторів показали точність гіршу за запропонований варіант.

3.4.2 Приклад 2. Класифікація.

В цьому прикладі були використані найсучасніші архітектури НМ, а саме згорткові НМ. Була вирішена сучасна актуальна задача класифікації зображень. Використовувався датасет Fashion MNIST (рис. 3.13).

Рис. 3.13 Приклад зображень датасету Fashion MNIST

Датасет містить 70000 зображень розміром 28x28 пікселів. 60000 було використано на тренувальну вибірку, з них 10% на валідаційну, а 10000 на тестову вибірку. Попередньо кожне зображення було пронормовано. Класифікація проводиться в 10 класів: t-shirt/top, trouser, pullover, dress, coat, sandal, shirt, sneaker, bag, ankle boot.

3.4.2.1 Архітектура ансамблю на основі оцінки індивідуальних вкладів

В цій реалізації використовувались 5 моделей згорткових НМ різної архітектури. Далі був застосований стекінг для об'єднання результатів. В якості мета-моделі було застосоване дерево прийняття рішень.

Розглянемо використані згорткові архітектури.

1. Мережа з одним згортковим шаром

Топологія мережі представлена на зображенні (рис. 3.14).

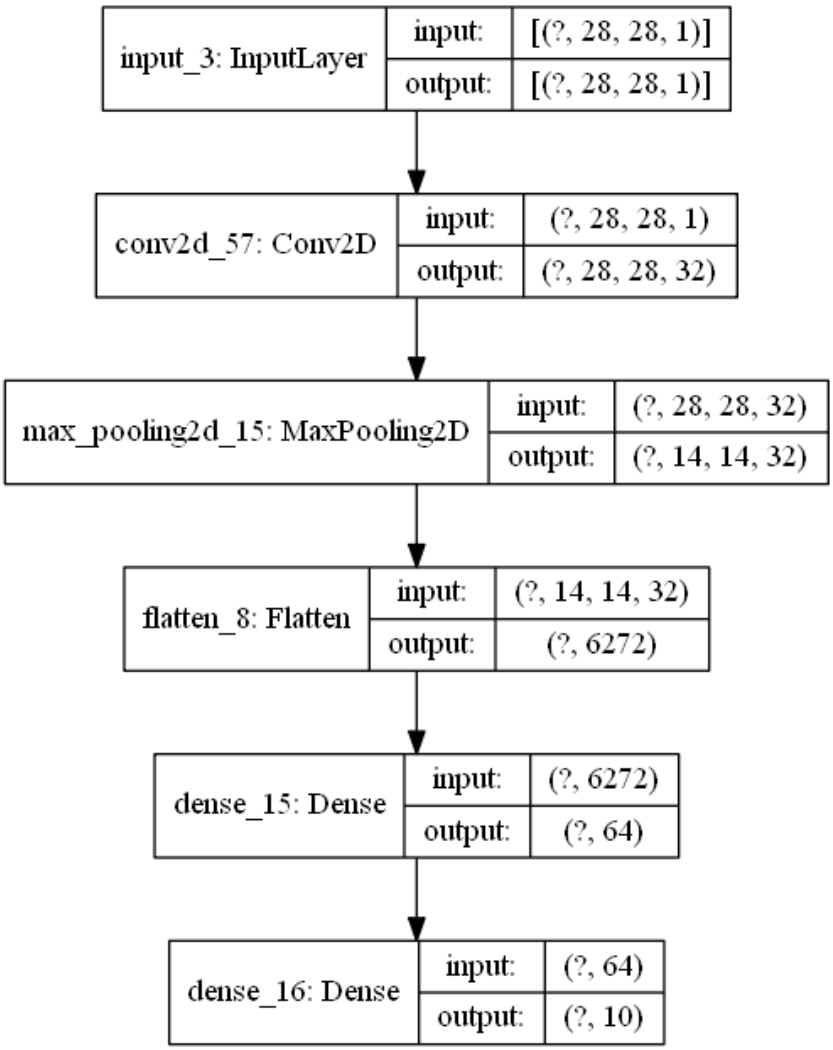


Рис. 3.14 Топологія НМ з одним згортковим шаром

Використовувались згортки розміром 3x3 з кількістю фільтрів – 32. Шар max pooling'у має розмір вікна 2x2. Оптимізатор – adam. 10 епох навчання.

2. Мережа з трьома згортковими шарами

Топологія представлена на зображенні (рис. 3.15).

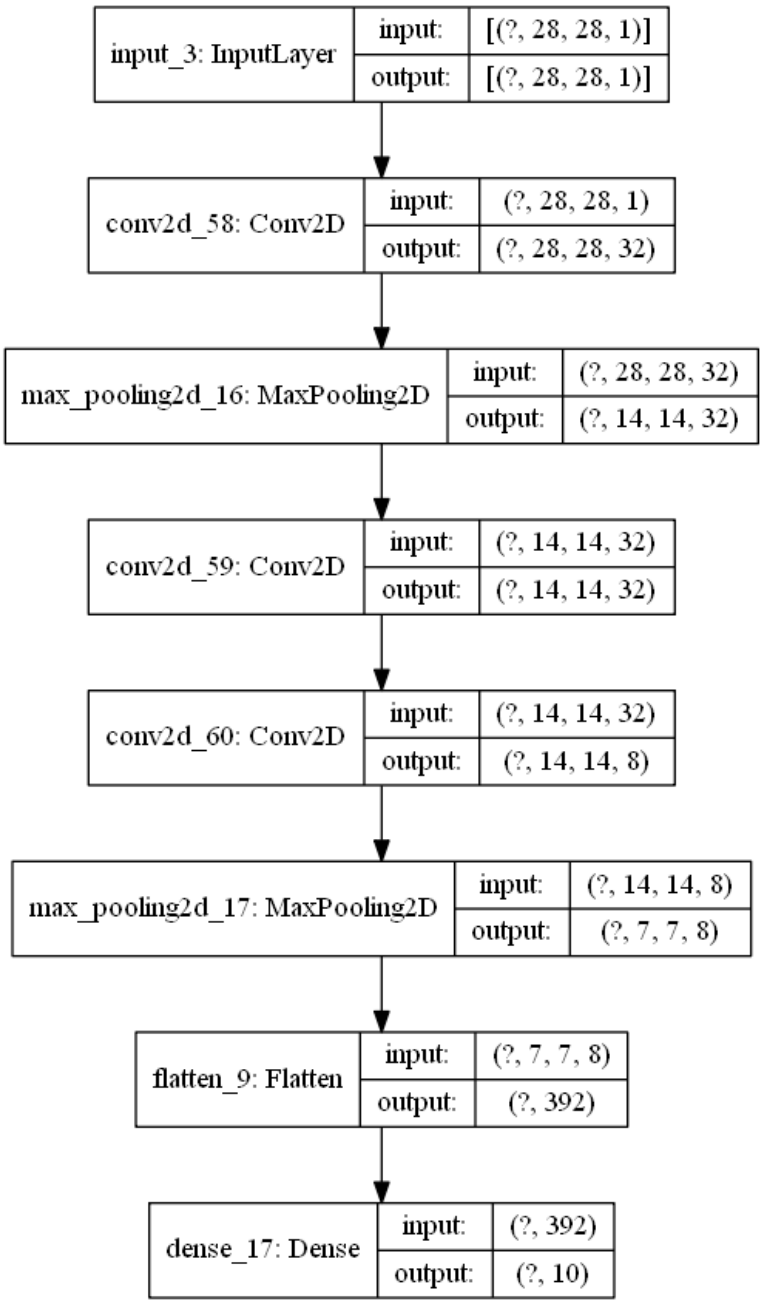


Рис. 3.15 Топологія НМ з трьома згортковими шарами

Використовувались згортки розміром 3x3 з кількістю фільтрів – 32, 32 і 8. Шар max pooling'у має розмір вікна 2x2. Оптимізатор – adam. 10 епох навчання.

3. ConvPool-CNN-C

Топологія моделі наведена в Додатку В.

4. ALL-CNN-C

Топологія моделі наведена в Додатку Г.

5. Network In Network CNN

Топологія моделі наведена в Додатку Ґ.

Результати навчання всіх згорткових класифікаторів зведені в таблицю (табл. 3.10).

Таблиця 3.10. Точність рішення згорткових класифікаторів

Класифікатор	Тренувальна точність	Тестова точність
One Layer CNN	95.18%	90.69%
Three Layers CNN	92.11%	90.39%
ConvPool-CNN-C	97.98%	91.93%
ALL-CNN-C	97.11%	91.74%
Network In Network CNN	90.66%	90.02%

Використаємо формулу (3.5) для знаходження ІВ кожної мережі на тренувальній множині (табл. 3.11).

Таблиця 3.11. Індивідуальний внесок окремих мереж

Класифікатор	ІВ
One Layer CNN	3373
Three Layers CNN	1921
ConvPool-CNN-C	12455
ALL-CNN-C	9169
Network In Network CNN	731

Об’єднаємо все мережі в мажоритарний ансамбль (табл. 3.12).

Таблиця 3.12. Точність рішення повного мажоритарного ансамблю

Тренувальна точність	Тестова точність
97.65%	93.36%

Можна помітити, що при звичайному мажоритарному голосуванні точність вже стала більшою.

Наступним етапом було застосування зменшення ансамблю на основі ІВ та побудова стекінгу з мета-моделлю на основі дерева прийняття рішень.

В скорочений ансамбль були взяті моделі з найвищими показниками ІВ:

– ConvPool-CNN-C;
– ALL-CNN-C;
– One Layer CNN.

В якості мета-моделі було використано глибоке дерево прийняття рішень з глибиною 15.

Повна топологія стекінг ансамблю наведена в Додатках Є і Ж. Оскільки дерево вийшло дуже широким, то представлено тільки його частину.

Результати навчання і вирішення задачі цим ансамблем представлені в таблиці (табл. 3.13).

Таблиця 3.13. Точність скороченого стекінг ансамблю

Тренувальна точність	Тестова точність
96.6%	97.14%

Можна помітити, що поєднання state-of-the-art архітектур дозволяє значно підвищити точність класифікації. В цьому і є заслуга ансамблів – поєднання різних незалежних топологій для різностороннього опису даних.

3.4.2.2 Інші архітектури ансамблів

Були використані наступні архітектури ансамблів для порівняння.

– Random Forest Classifier;
– AdaBoost з деревами прийняття рішень;
– bagging з деревами прийняття рішень;
– bagging з 5 невеликими згортковими мережами.

Результати представлені в таблиці (табл. 3.14).

Таблиця 3.14. Точність різних архітектур ансамблів

Ансамбль	Тренувальна точність	Тестова точність
Random Forest	97.32%	87.15%
AdaBoost	54.96%	54.25%
Bagging	94.41%	83.85%
CNN bagging	94.96%	91.39%

По результатам видно, що запропонована структура набагато краще впоралась із завданням, ніж інші ансамблеві методи.

3.4.3 Приклад 3. Апроксимація.

Вирішується задача передбачення вартості будинків у Кінг-Каунті, штат Вашингтон, США. Набір даних складається з історичних даних про будинки, продані в період з травня 2014 року по травень 2015 року.

Кількість зразків в наборі – 21613. Кожен об'єкт має 18 ознак, представлених дійсними числами більше нуля, і одне значення вартості (табл. 3.15). Для тестової вибірки обрані 25% набору даних, відповідно, 75% – для навчання.

Таблиця 3.15. Приклад вибірки даних

Назва ознаки	#1	#2	#3
bedrooms	3	2	3
bathrooms	1	1	2.25
sqft_living	1180	770	2570
sqft_lot	5650	10000	7242
floors	1	1	2
waterfront	0	0	0
view	0	0	0
condition	3	3	3
grade	7	6	7
sqft_above	1180	770	2170
sqft_basement	0	0	400
year_built	1955	1933	1951
year_renovated	0	0	1991
zipcode	98178	98928	98125
latitude	47.5112	47.7379	47.721
longtitude	-122.2570	-122.2330	-122.319
sqft_living15	1340	2720	1690
sqft_lot15	5650	8062	7639
Вартість	221900	180000	538000

Розглянемо загальну структуру ансамблю. Було використано 4 базові моделі-апроксиматори.

1. Stochastic Gradient Descent Regressor

Лінійна модель, яка навчається мінімізацією регульованих емпіричних втрат за допомогою SGD. Перед навчанням дані були стандартизовані. Кількість ітерацій навчання було поставлено рівним 1000.

2. Ridge регресія

Ridge регресія вирішує деякі проблеми звичайного методу найменших квадратів шляхом накладення штрафу за розміром коефіцієнтів. Ridge коефіцієнти мінімізують залишкову суму квадратів зі штрафом:

$$\min_w \|Xw - y\|_2^2 + \alpha \|w\|_2^2 \qquad (3.9)$$

де α – параметр складності, контролює кількість усадки: чим більша величина, тим більша кількість усадки і, отже, коефіцієнти стають більш стійкими до колінеарності; w – коефіцієнти моделі, на які накладається штраф.

3. K-neighbors regressor

Регресія на основі алгоритму k найближчих сусідів може використовуватися в тих випадках, коли мітки даних є неперервними, а не дискретними змінними. Мітка, призначена для точки запиту, обчислюється, виходячи із середнього значення міток найближчих сусідів.

Значення прогнозується локальною інтерполяцією значень, пов'язаних з найближчими сусідами в навчальному наборі.

Було створено 5 кластерів.

4. Decision Tree Regressor

Decision Tree Regressor зазвичай використовує середню квадратичну помилку (MSE), щоб вирішити розділити вузол на два або більше підвузлів.

Припустимо, ми робимо двійкове дерево, алгоритм спочатку вибере значення та розділить дані на два підмножини. Для кожного підмножини він буде обчислювати MSE окремо. Дерево вибирає значення з результатами найменшого значення MSE (рис. 3.16).

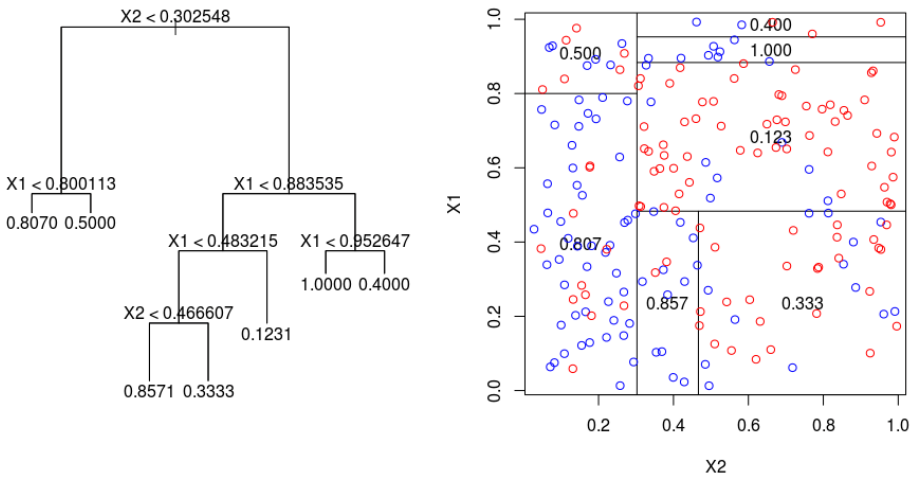


Рис. 3.16 Приклад Decision tree regressor

Результати навчання окремих апроксиматорів наведені в таблиці (табл. 3.16).

Таблиця 3.16. Результати навчання окремих апроксиматорів, оцінка MAPE

Апроксиматор	Тренувальна помилка	Тестова помилка
Linear SGD	26.36%	25.68%
Decision Tree	15.63%	17.56%
Ridge	25.58%	24.88%
K-neighbors	25.59%	31.34%

Далі була побудована архітектура стекінгового ансамблю з використанням декількох мета-моделей для порівняння. Були використані наступні агрегуючі апроксиматори:

– лінійна регресія;
– дерево прийняття рішень для регресії;
– МГУА мережа.

Для МГУА мережі ступінь поліному був обраний 3, а кількість входів в нейрони – 2.

Результати створення і навчання таких ансамблів наведені в таблиці (табл. 3.17).

Таблиця 3.17. Результати навчання стекінг ансамблів з різними мета-моделями

Мета-модель	Тренувальна помилка	Тестова помилка
Linear regression	15.19%	16.72%
Decision Tree Regression	14.4%	16.13%
МГУА	13.4%	15.66%

Результати показали, що в загальному використання ансамблю зменшило помилку апроксиматорів на датасеті. А використання МГУА, як мета-моделі, за показниками MAPE є найбільш доцільним і кращим. Загальна топологія ансамблю наведена на зображенні (рис. 3.17).

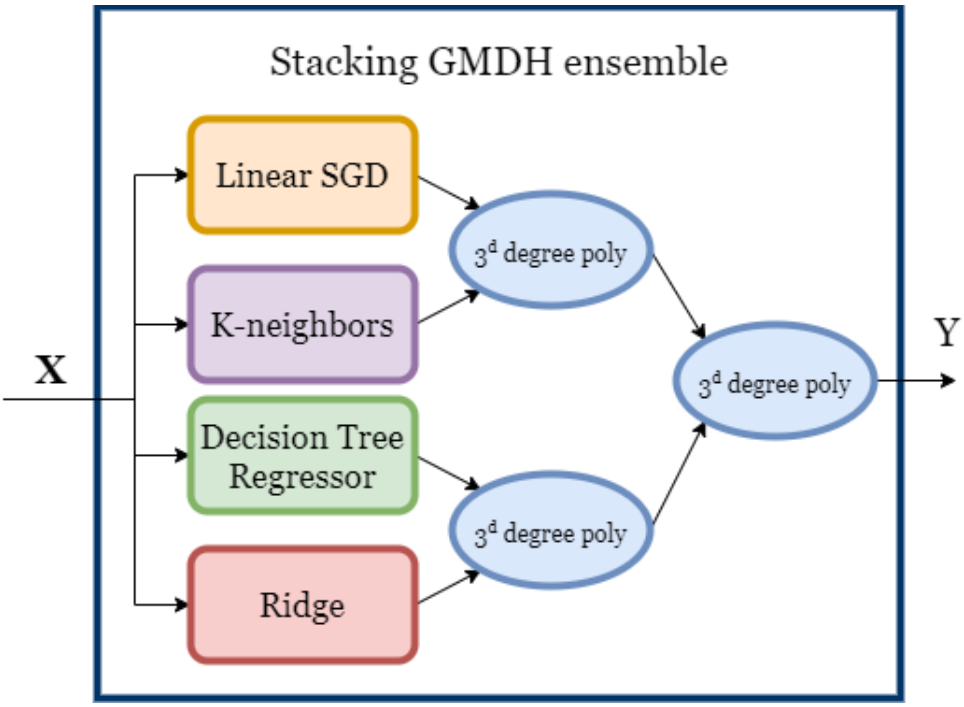


Рис. 3.17 Фінальна топологія стекінг ансамблю з МГУА

ВИСНОВКИ ДО РОЗДІЛУ

Були запропоновані, розглянуті та досліджені методи зменшення та синтезу ансамблевої топології: зменшення на основі ІВ та скорочення, синтез стекінг ансамблю з агрегуючою МГУА мережою. Були детально описані використані архітектури моделей НМ та проведені практичні дослідження: два приклади для вирішення задачі класифікації і один приклад для задачі апроксимації.

Загальний аналіз результатів показав, що використання ансамблевих топологій дає показники точності, які перевершують відповідні показники окремих мереж, що дозволяє значно покращити результати вирішення поставлених задач. Це знову підтверджує значимість ансамблевих методів в сфері машинного навчання.

В першому прикладі завдяки використанню ІВ та методу скорочення розмір ансамблю був зменшений майже вдвічі, а використання зважування, як модуля поєднання, дозволило підвищити точність класифікації. Отримані

результати показують, що використання зваженого скороченого ансамблю різних класифікаторів перевершує інші популярні архітектури (беггинг, бустінг, випадкові ліси) за показниками точності.

В другому прикладі були використані сучасні згорткові архітектури та процедура стекінгу на основі дерева прийняття рішень. Ансамбль був скорочений на основі ІВ, що зменшило його розмір з 5 до 3 моделей. Звичайне мажоритарне голосування вже показувало результати точності перевершуючі окремі згорткові мережі. А використання стекінгу з деревом прийняття рішень значно підвищило фінальну точність загального класифікатора. Інші архітектури ансамблів залишилися далеко позаду.

Третій приклад на апроксимацію підтвердив умовиводи та ідеї щодо МГУА мережі в стекінг топології. В якості окремих апроксиматорів були використані доволі прості та швидкі моделі, їх поєднання в ансамбль дозволило значно зменшити загальну помилку. Були розглянуто декілька популярних мета-моделей, але в загальному результаті МГУА мережа показала найкращий результат.

Загальним висновком є підтвердження описаних вище існуючих та власних ідей щодо структурно-параметричного синтезу НМ на основі ансамблевої топології.

РОЗДІЛ 4 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Опис використаних пакетів

Сучасні технології дозволяють працювати з величезними мережами, як із простими елементами для побудови чогось більшого, наприклад, цегли LEGO. Рівень абстракції сучасного програмного забезпечення дуже високий, все знаходиться на інтуїтивно зрозумілому та зрозумілому рівні, тому практична реалізація синтезу НМ – досить легке завдання.

Основний внесок в створення сучасних бібліотек машинного навчання, а особливо програмування нейронних мереж і глибокого навчання, внесли окремі інженери і команди корпорації Google. Вони створили кілька основних бібліотек з відкритим вихідним кодом для спрощення створення практичних проєктів з використанням НМ. Також чималий внесок вносять фахівці компанії Facebook.

Більшість існуючих бібліотек написані і використовуються під крос-платформний багатофункціональний мова програмування Python. Програмне забезпечення даної роботи цілком написано на цій мові.

Основні бібліотеки для створення НМ на даний момент, які також були використані в даній роботі, наступні:

1. Tensorflow [44]

TensorFlow – це безкоштовна бібліотека програмного забезпечення з відкритим кодом для обробки даних та диференційованого програмування для різних завдань. Це символьна математична бібліотека, що також використовується для машинного навчання, нейронних мереж. Вона використовується як для досліджень, так і для комерційних систем.

TensorFlow – це система другого покоління Google Brain. Версія 1.0.0 була передана в оренду 11 лютого 2017 року. Хоча реалізація посилань працює на одних пристроях, TensorFlow може працювати на декількох процесорах та графічних процесорах (з додатковими розширеннями CUDA та SYCL для

обчислення загального призначення на одиницях обробки графіки). TensorFlow доступний на 64-бітних Linux, macOS, Windows та мобільних обчислювальних платформах, включаючи Android та iOS.

Його гнучка архітектура дозволяє легко розгортати обчислення на різних платформах (процесори, графічні процесори, TPU), а також від настільних комп'ютерів до кластерів серверів, до мобільних та крайових пристроїв.

Обчислення TensorFlow виражаються у вигляді графіків стану даних. Назва TensorFlow походить від операцій, які такі нейронні мережі виконують на багатовимірних масивах даних, які називають тензорами.

Дана бібліотека дозволяє створювати і оперувати моделями навчання в парадигмі ООП, що є дуже зручною абстракцією. Також Tensorflow дозволяє просунутим дослідникам використовувати низькі абстракції, наприклад тензори, сесії, змінні для створення більш гнучких і призначених для користувача моделей. Однак для багатьох досліджень вони мають уже підготовлені класи шарів або навіть готових мереж, які досить легко імпортуються і впроваджуються.

2. Keras [45]

Keras – бібліотека нейронних мереж з відкритим кодом, написана на Python. Вона може працювати над TensorFlow, Microsoft Cognitive Toolkit, R, Theano або PlaidML. Розроблена для швидких експериментів з глибокими нейронними мережами, фокусується на зручності, модульності та розширюваності.

У 2017 році команда Google TensorFlow вирішила підтримати Keras в основній бібліотеці TensorFlow. Було пояснено, що Keras замислювався як інтерфейс, а не окремий фреймворк машинного навчання. Він пропонує більш інтуїтивно зрозумілий набір абстракцій, що спрощує розробку моделей глибокого навчання незалежно від використовуваних обчислювальних програм.

Корпорація Майкрософт додала до Keras також сервер CNTK, який доступний на версії CNTK v2.0.

Keras містить численні реалізації часто використовуваних нейронних мережних будівельних блоків, таких як шари, цілі, функції активації, оптимізатори та безліч інструментів для полегшення роботи з зображеннями та текстовими даними, щоб спростити необхідний код кодування для написання глибокого нейронного мережевого коду. Код розміщений на GitHub, а форуми підтримки спільноти включають сторінку питань GitHub та канал Slack.

Окрім стандартних нейронних мереж, Keras має підтримку згорткових і рекурентних нейронних мереж. Він підтримує інші загальні шари, такі як дропаут, нормалізація та об'єднання.

Keras дозволяє користувачам продукувати глибокі моделі на смартфонах (iOS та Android), в Інтернеті або на віртуальній машині Java. Це також дозволяє використовувати розподілене навчання моделей глибокого навчання на кластерах одиниць графічної обробки (GPU) та тензорних процесорів (TPU), головним чином спільно з CUDA.

Завдяки даній бібліотеці дійсно можна оперувати мережами або шарами, як блоками конструктора. Тут є великий рівень абстракцій ООП. Основними оперованими класами є моделі НМ, шари, нейрони, оптимізатори. На малюнку 4.1 наведено приклад досить складної моделі, описаній в високих абстракціях цієї бібліотеки.

```
from tensorflow import keras
from tensorflow.keras import layers

# Instantiate a trained vision model
vision_model = keras.applications.ResNet50()

# This is our video encoding branch using the trained vision_model
video_input = keras.Input(shape=(100, None, None, 3))
encoded_frame_sequence = layers.TimeDistributed(vision_model)(video_input)
encoded_video = layers.LSTM(256)(encoded_frame_sequence)

# This is our text-processing branch for the question input
question_input = keras.Input(shape=(100,), dtype='int32')
embedded_question = layers.Embedding(10000, 256)(question_input)
encoded_question = layers.LSTM(256)(embedded_question)

# And this is our video question answering model:
merged = keras.layers.concatenate([encoded_video, encoded_question])
output = keras.layers.Dense(1000, activation='softmax')(merged)
video_qa_model = keras.Model(inputs=[video_input, question_input],
                             outputs=output)
```

Рис. 4.1 Приклад моделі Keras

3. PyTorch [46]

PyTorch – це бібліотека машинного навчання з відкритим кодом, заснована на бібліотеці Torch, яка використовується для таких додатків, як комп'ютерне бачення та обробка природних мов, розроблена насамперед лабораторією AI Research Facebook (FAIR). Це безкоштовне та відкрите програмне забезпечення, що випускається під ліцензією Modified BSD. Хоча інтерфейс Python є більш якісним та основним напрямком розвитку, PyTorch також має інтерфейс мови C++.

На PyTorch побудовано ряд програм програм Deep Learning, включаючи Pyro Uber, Трансформери HuggingFace та Catalyst.

PyTorch надає дві функції високого рівня:

– тензорні обчислення (наприклад, NumPy) з сильним прискоренням за допомогою графічних процесорних блоків (GPU);
– глибокі нейронні мережі, побудовані на основі стрічкової системи автоматичної диференціації.

Модулі:

– Autograd module

PyTorch використовує метод, який називається автоматичною диференціацією. Рекордер записує виконані операції, а потім він відтворює його назад для обчислення градієнтів. Цей метод особливо потужний при побудові нейронних мереж для економії часу в одну епоху шляхом обчислення диференціації параметрів при прямому проході.

– Модуль Optim

torch.optim – це модуль, який реалізує різні алгоритми оптимізації, що використовуються для побудови нейронних мереж. Більшість часто використовуваних методів вже підтримуються, тому немає необхідності будувати їх з нуля.

– Модуль NN

Автоград PyTorch дозволяє легко визначати обчислювальні графіки та приймати градієнти, але необроблений автоград може бути трохи занадто низьким для визначення складних нейронних мереж. Тут може допомогти nn модуль.

На зображенні (рис. 4.2) показаний приклад створення і навчання простої НМ.

```
class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        """
        In the constructor we instantiate two nn.Linear modules and assign them as
        member variables.
        """
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        """
        In the forward function we accept a Tensor of input data and we must return
        a Tensor of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Tensors.
        """
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred


# N is batch size; D_in is input dimension;
# H is hidden dimension; D_out is output dimension.
N, D_in, H, D_out = 64, 1000, 100, 10

# Create random Tensors to hold inputs and outputs
x = torch.randn(N, D_in)
y = torch.randn(N, D_out)

# Construct our model by instantiating the class defined above
model = TwoLayerNet(D_in, H, D_out)

# Construct our loss function and an Optimizer. The call to model.parameters()
# in the SGD constructor will contain the learnable parameters of the two
# nn.Linear modules which are members of the model.
criterion = torch.nn.MSELoss(reduction='sum')
optimizer = torch.optim.SGD(model.parameters(), lr=1e-4)
for t in range(500):
    # Forward pass: Compute predicted y by passing x to the model
    y_pred = model(x)

    # Compute and print loss
    loss = criterion(y_pred, y)
    if t % 100 == 99:
        print(t, loss.item())

    # Zero gradients, perform a backward pass, and update the weights.
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Рис. 4.2 Приклад використання PyTorch

4. SciKit-learn [47]

Scikit-learn (також відомий як sklearn) – це безкоштовна бібліотека машинного навчання для мови програмування Python. У ній представлені різні алгоритми класифікації, регресії та кластеризації, включаючи support vector

machines, random forests, gradient boosting, k-means and DBSCAN, і розроблений для взаємодії з числовими та науковими бібліотеками Python NumPy та SciPy.

Scikit-learn - це бібліотека в Python, яка надає безліч неконтрольованих та контрольованих алгоритмів навчання. Він побудований на основі тієї технології NumPy, pandas та Matplotlib.

Функціонал, який надає scikit-learn, включає:

– regression, including Linear and Logistic Regression;
– classification, including K-Nearest Neighbors;
– clustering, including K-Means and K-Means++;
– model selection;
– preprocessing, including Min-Max Normalization.

На зображенні (рис. 4.3) зображено приклад предобробки даних з використанням scikit-learn.

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>> X_scaled = preprocessing.scale(X_train)

>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Рис. 4.3 Приклад використання scikit-learn

4.2 Структура програмного забезпечення

Програмне забезпечення, створене в цій роботі, покликане вирішити наступні задачі:

– використання єдиного програмного інтерфейсу для керування і побудови гібридних моделей машинного навчання з реалізацією на будь-якій прикладній технології (tensorflow, keras, pyTorch, sklearn, власна реалізація). Тобто при підтримці створеного програмного інтерфейсу можливий легкий

синтез будь-якої топології (зокрема, ансамблевої) будь-яких моделей, використовуючи розроблені алгоритми та абстракції;

– створення дослідницького модуля з реалізованими прикладними НМ та алгоритмами для швидкого синтезу потрібних топологій та вирішення прикладних задач;
– розробка клієнтського інтерфейсу командного рядка для побудови і використання найкращої ансамблевої моделі, описуючої переданий набір даних.

Основна частина програмного забезпечення та розрахунків цієї роботи було створено на крос-платформній мові програмування Python в середовищі Jupyter Notebook платформи Anaconda з використанням бібліотек, описаних вище.

Програмне забезпечення проєкту має наступну структуру (рис. 4.4):

– модуль основних розроблених інтерфейсів та алгоритмів моделей машинного навчання, може імпортуватися для використання в дослідницьких цілях або для побудови прикладних моделей;
– модуль допоміжних утиліт для обробки наборів даних, зберігання та завантаження моделей і т.п.;
– клієнтський інтерфейс командного рядка;
– файл приклад з налаштуванням, навчанням і розрахунками моделей для прикладу 1;
– аналогічний файл для прикладу 2 з генерацією зображень моделей і ансамблю;
– такий самий файл для прикладу 3.

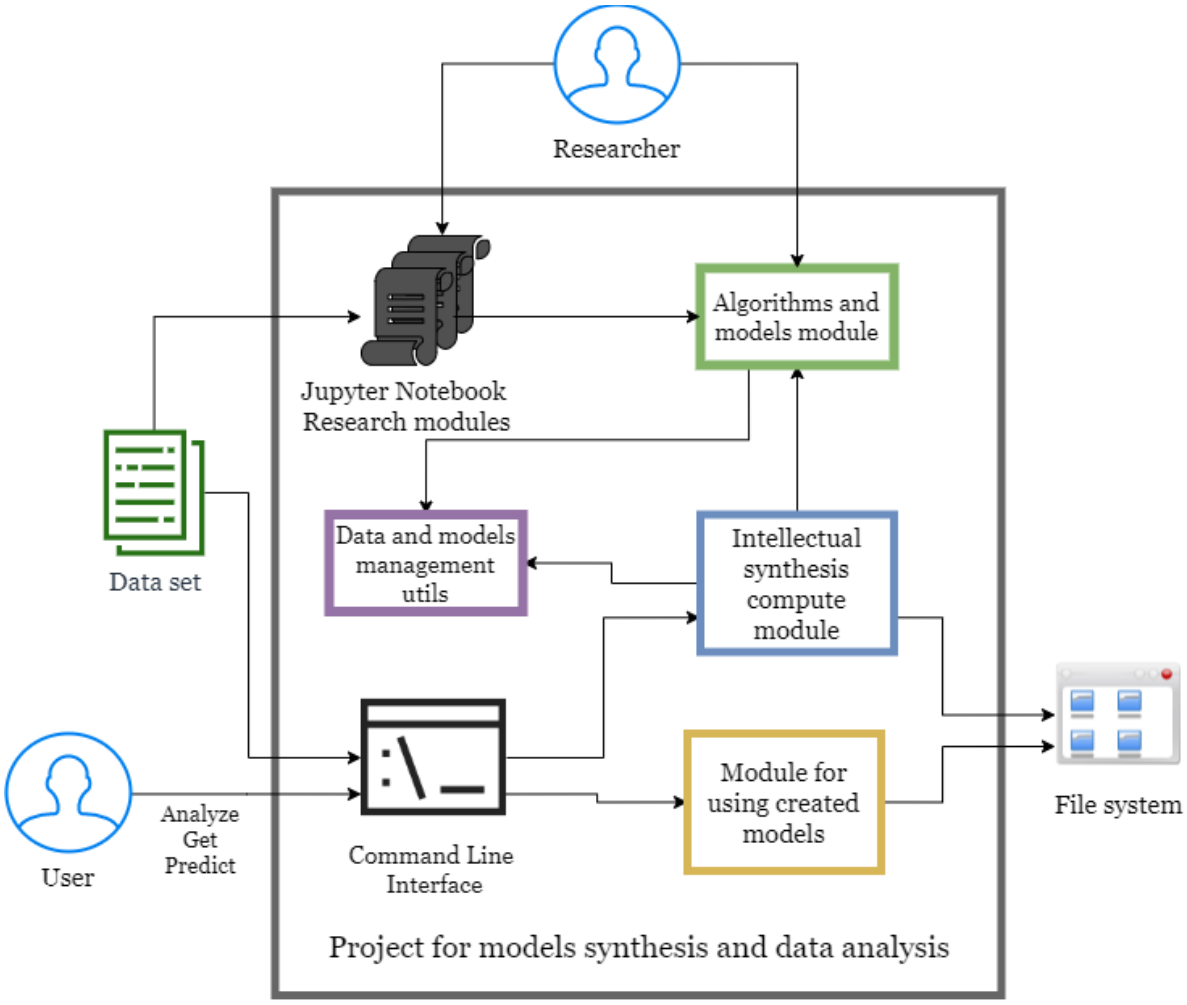


Рис. 4.4 Загальна структура проєкту

Перший модуль містить власні реалізації наступних алгоритмів та НМ (табл. 4.1).

Таблиця 4.1. Опис реалізованих архітектур

Модель/Алгоритм	Назва класу в пакеті	Використано бібліотеку
Multilayer Perceptron	MLP	keras
МГУА	GMDH	Numpy, scikit-learn
Мережа Кохонена	KohonenNetwork	numpy
Радіально-базисна мережа	RBF	Keras with tensorflow backend
Ймовірнісна НМ	PNN	numpy
Naïve Bayes Classifier	NaiveBayes	numpy
NefClassM	NefClassM	PyTorch
Takagi-Sugeno-Kang	TSK	PyTorch
Гібрид TSK та МГУА	TSKGMDH	–
Мережа зустрічного поширення	CPN	numpy
Алгоритм розрахунку індивідуального вкладу ансамблю	IC	numpy
Загальні CNN моделі	ConvNet	Keras with tensorflow backend
ConvPool-CNN-C	ConvPoolCNN	Keras with tensorflow backend
ALL-CNN-C	AllCNNC	Keras with tensorflow backend
Network In Network CNN	NiNCNN	Keras with tensorflow backend

Розроблені модулі є універсальними і можуть використовуватися, як окремий пакет НМ для багатьох інших проєктів, систем, архітектур (рис. 4.5): клієнт-серверна архітектура, native інтерфейси, використання в дослідженнях в Jupyter Notebook і т. д.

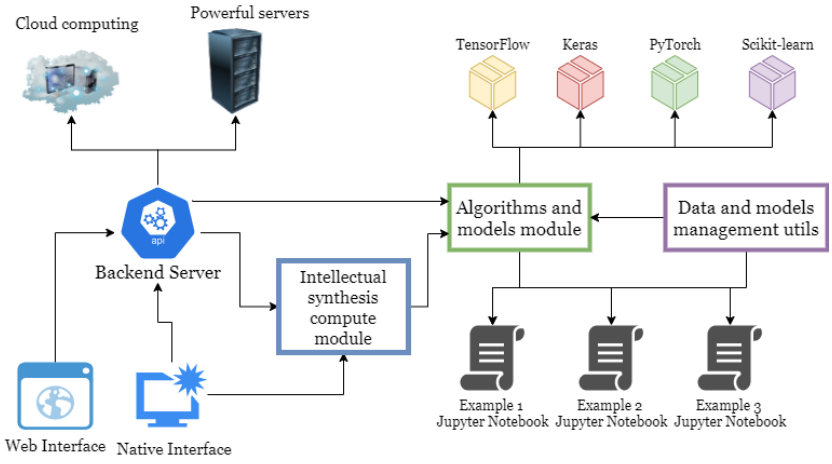


Рис. 4.5 Архітектура можливого використання розроблених модулей

Логічним використанням пакету є створення бекенд архітектури на потужних серверах, де б могли тренуватися і зберігатися важкі моделі, а також вести базу даних моделей, результатів, датасетів, для доступу має бути створене API.

Реалізація описаних вище архітектур НМ була зроблена в парадигмі об’єктно орієнтованого програмування. Був розроблений єдиний інтерфейс класу для кожної моделі (див. Додаток Д). Використання єдиного інтерфейсу дозволяє уніфікувати моделі та абстрагуватись від використовуваних бібліотек, алгоритмів. Додаток Е ілюструє приклад реалізації одної з прикладних НМ в створеному програмному інтерфейсі.

Розроблений інтерфейс командного рядка дозволяє користувачу завантажити свій датасет для створення моделі ансамблю, яка б максимально описувала створений датасет і робила б точні прогнози для нових даних. Далі користувач може робити нові прогнози завдяки інтерфейсу для нового файлу з даними посилаючись на створену модель. Діаграми послідовностей для описаних двох процесів представлена в Додатку З і И.

Створений проєкт дозволяє зберегти синтезовані та натреновані моделі в декілька форматів для подальшої передачі або простого зберігання і завантаження. Можливі формати зберігання та завантаження моделей:

– yaml;
– json;
– h5 (для вагів мереж бібліотеки Keras);
– pickle (бінарний формат Python’у).

Використання модулів проєкту налаштоване на роботу в бекенд системах, наприклад, можливе використання пакету з веб-фреймворком Flask для побудови серверного API.

4.3 Контрольний приклад

В якості контрольного прикладу розробленого ПЗ та алгоритмів був створений інтелектуальний модуль реального часу рухомої робототехнічної системи. Робот має 24 вбудовані датчики на кожні 15° обзору (рис. 4.6).

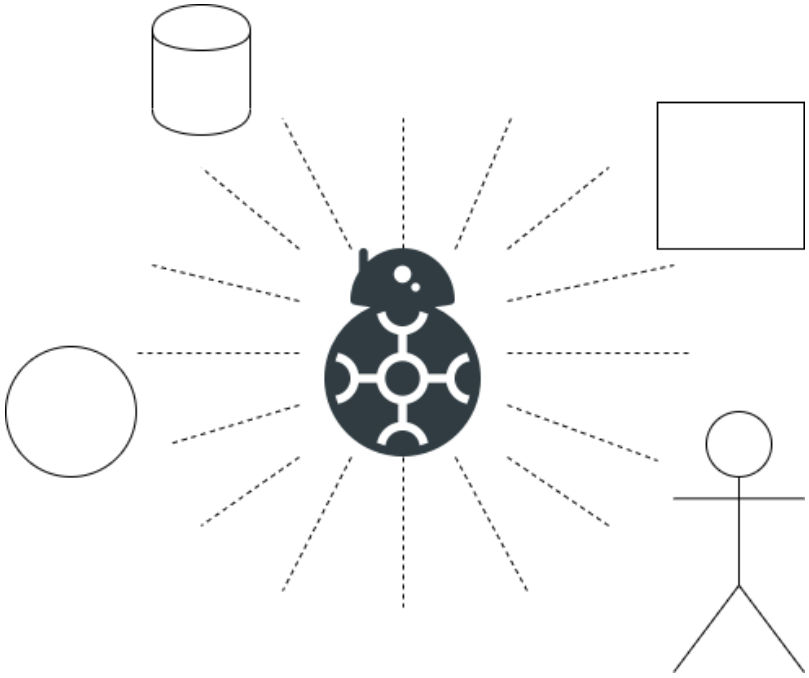


Рис. 4.6 Приклад роботу з багатьма датчиками

Кожну відмітку часу робот має проаналізувати дані з кожного датчику і зробити рух. Кожен датчик дає дійсне значення від 0 до 5 присутності перешкоди. Можливі рухи: просування вперед, поворот вліво, поворот направо, сильний поворот вправо.

Був використаний зібраний експериментальний датасет для побудови інтелектуальної моделі прогнозування наступного руху робота на основі поточних даних датчиків. Модель була побудована з використанням створеного ПЗ на ансамблевій топології.

Зібраний датасет був завантажений в csv файл, де останні значення – це клас руху (рис. 4.7).

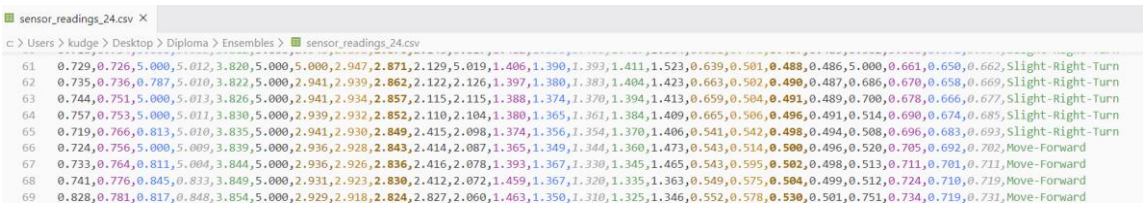


Рис. 4.7 Частина датасету в форматі csv

Файл датасету був переданий як аргумент інтерфейсу з командною build для побудови ансамблю та збереження ансамблю (рис. 4.8).

```
(dev) C:\Users\kudge\Desktop\Diploma\Ensembles>python ensemple.py build sensor_readings_24.csv --modelname robot
Check dataset...
Ok
Building model...
Problem: classification
Selecting models: NaiveBayse, RBFN, MLP, DecisionTree, SVC...
Selected: DecisionTree, KNearestNeighbors, SVC, MLP
Pruning: removed SVC
Build ensemble...
Train accuracy: 96.06%
Test accuracy: 95.45%
Model `robot` saved
Finished.
```

Рис. 4.8 Приклад створення і збереження ансамблю для файлу датасету

Система досить добре впоралася із завданням. Створений ансамбль правильно класифікує 96.06% тренувальних значень датчиків і 95.45% тестових значень.

Далі можливе використання створеної моделі для прогнозування нового руху для нових отриманих даних з датчиків. Потрібно завантажити дані в файл csv та передати до інтерфейсу з командою predict та назвою моделі (в даному випадку robot) (рис. 4.9).

```
(dev) C:\Users\kudge\Desktop\Diploma\Ensembles>python ensemple.py predict new_values.csv --modelname robot
Check data...
Ok
Check model...
Ok
Loading model...
Making prediction...
Class: Sharp-Right-Turn
```

Рис. 4.9 Приклад прогнозування нового руху

Для нових значень датчиків модель видала рішення крутого повороту направо. Можливо також завантажити передбачення в файл (рис. 4.10).

```
(dev) C:\Users\kudge\Desktop\Diploma\Ensembles>python ensemple.py predict new_values.csv --modelname robot --tofile move
Check data...
Ok
Check model...
Ok
Loading model...
Making prediction...
Class: Move-Forward
Prediction saved to `move` file
```

Рис. 4.10 Приклад збереження спрогнозованого руху до файлу

Вміст файлу представлений на зображенні (рис. 4.11).

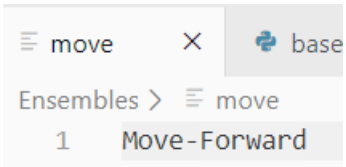


Рис. 4.11 Вміст файлу з наступним вирахуваним рухом

ВИСНОВКИ ДО РОЗДІЛУ

Була розроблена система створення моделей машинного навчання ансамблевої топології та аналізу даних на мові програмування Python на основі нейронних мереж та ансамблевої топології. Були описані використані найсучасніші бібліотеки, завдяки яким тренування мереж та налаштування алгоритмів здійснюється швидко та зрозуміло: tensorflow, keras, pytorch, scikit-learn. Програмне забезпечення побудоване на парадигмі об’єктно-орієнтованості. Був розроблений єдиний програмний інтерфейс для побудови моделі незалежно від реалізації та використаних бібліотек та незалежно від топології моделі – звичайна НМ, модуль, ансамбль, що дозволяє легко маніпулювати, об’єднувати та використовувати будь-які великі синтезовані структури.

Розроблений модуль алгоритмів і моделей готовий до використання в багатьох різноманітних системах: від власних досліджень в системі Jupyter Notebook чи звичайних Python-файлах до роботи на потужних серверах для синтезу, тренування та зберігання великих моделей.

Був розроблений інтерфейс командного рядка для швидкого і прозорого створення моделей і продукування передбачень.

В якості контрольного прикладу з використанням інтерфейсу керування моделями була створена ансамблева структура класифікатора реального часу для керування роботехнічною системою на основі даних з 24 датчиків. Розроблена модель має велику точність класифікації і вирахування наступного кроку робота – 95.45% на тестовому наборі і готова до впровадження до робототехнічної системи в якості інтелектуального модуля.

ВИСНОВКИ

Аналіз сфери машинного навчання показав, що гібридизація та синтез різних моделей, в тому числі і нейронних мереж, є популярним та ефективним сучасним рішенням до створення прогнозуючих та обробляючих структур. Були розглянуті різні підходи до модернізації та об'єднання сучасних архітектур: модифікація структури штучного нейрону, ускладнення вигляду нелінійності (вейвлони), додавання зв'язків пам'яті (LSTM, GRU), побудова модульних топологій (Encoder-Decoder; на основі мереж неконтрольованого навчання) та потужних ансамблевих методів. Було досліджено, що саме останні ансамблеві методи в комбінації з іншими гібридними структурами можуть дати великий поштовх при обробці даних.

Були опрацьовані різні підходи до синтезу ансамблів: беггинг, бустинг, стекінг, random forests, алгоритми градієнтного бустингу, зваженого поєднання моделей. Були вивчені переваги кожного методу, можливості використання та модифікації. Був розроблений та досліджений алгоритм поєднання на основі критеріїв різноманітності та індивідуального внеску кожної моделі в фінальну ансамблеву структуру для вирішення задач класифікації. Також були опрацьовані методи скорочення ансамблю для підтримки балансу розміру, швидкості та точності. Практичні приклади з використанням сучасних архітектур нейронних мереж підтвердили правильність та переваги теоретичних досліджень. Була запропонована технологія використання МГУА мережі в якості мета-агрегуючої моделі в стекінг архітектури для вирішення задач апроксимації. Практичний приклад на реальних даних показав, що розроблена структура має менші показники помилки, ніж інші розглянуті архітектури.

Була розроблена програмна система створення та керування моделями машинного навчання, ансамблевими та модульними структурами, аналізу даних на мові програмування Python. Були використані найсучасніші бібліотеки, завдяки яким тренування мереж та налаштування алгоритмів здійснюється

швидко та зрозуміло: tensorflow, keras, pytorch, scikit-learn. Програмне забезпечення побудоване на парадигмі об’єктно-орієнтованості та орієнтоване на клієнт-серверну архітектуру. Виконання розробленого пакету на потужних серверах значно пришвидшить розробку моделей. Був розроблений єдиний програмний інтерфейс для побудови моделі незалежно від реалізації та використаних бібліотек, та незалежно від топології моделі – звичайна НМ, модуль, ансамбль, що дозволяє легко маніпулювати, об’єднувати та використовувати будь-які великі синтезовані структури.

Розроблений модуль алгоритмів і моделей готовий до використання в багатьох різноманітних системах: від власних досліджень в системі Jupyter Notebook чи звичайних Python-файлах до роботи на потужних серверах для синтезу, тренування та зберігання великих моделей і доступу до них через графічний інтерфейс.

Був розроблений тестовий інтерфейс командного рядка для швидкого і прозорого створення моделей і продукування передбачень на власній обчислювальній машині.

В якості контрольного прикладу з використанням розробленого інтерфейсу керування моделями була створена ансамблева структура класифікатора реального часу для керування роботехнічною системою на основі даних з 24 датчиків. Розроблена модель має велику точність класифікації і вирахування наступного кроку робота: 95.45% на тестовому наборі і готова до впровадження в робототехнічну систему в якості інтелектуального модуля.

Загалом було проведено науково-дослідницьку роботу по використанню ансамблевих топології та гібридного синтезу і подальша практична програмна реалізація вивчених та розроблених архітектур та алгоритмів в єдину систему машинного навчання.

ПЕРЕЛІК ПОСИЛАНЬ

1 Google DeepMind Department [Електронний ресурс] – Режим доступу до ресурсу: https://deepmind.com/about/deepmind-for-google

2 Amazon AI Department [Електронний ресурс] – Режим доступу до ресурсу: https://www.amazon.jobs/en/teams/amazonai

3 Microsoft Research Lab – AI [Електронний ресурс] – Режим доступу до ресурсу: https://www.microsoft.com/en-us/research/lab/microsoft-research-ai/

4 Lai, K.K. An Integrated Data Preparation Scheme for Neural Network Data Analysis.IEEE Trans. Knowl.Data Eng.2006,18, 217–230. DOI: 10.1109/TKDE.2006.22

5 Zhang Y, Guo Q, Wang Jw. Big data analysis using neural networks. Adv Eng Sci. 2017;49:9–18. DOI: 10.15961/j.jsuese.2017.01.002

6 Liu W, Wang Z, Liu X, Zeng N, Liu Y, Alsaadi FE (2017) A survey of deepneural network architectures and their applications. Neurocomputing234:11–26. DOI: 10.1016/j.neucom.2016.12.038

7 Real-life neural networks applicaions [Електронний ресурс] – Режим доступу до ресурсу: https://www.smartsheet.com/neural-network-applications

8 Python Machine Learning, 2nd Edition. Sebastian Raschka, Vahid Mirjalili. p. 622, Packt Publishing Ltd. (September 20th, 2017).

9 Николенко С.,Кадурин А., Архангельская Е. Глубокое обучение. Погружение в мир нейронных сетей. - СПб.: Питер, 2018. - 480 с.

10 5 algorithms to train neural networks [Електронний ресурс] – Режим доступу до ресурсу: https://www.neuraldesigner.com/blog/5_algorithms_to_train_a_neural_network

11 Genetic Algorithms + Neural Networks = Best of Both Worlds [Електронний ресурс] – Режим доступу до ресурсу: https://towardsdatascience.com/gas-and-nns-6a41f1e8146d

12 Robbins, Herbert; Monro, Sutton. A Stochastic Approximation Method. doi:10.1214/aoms/1177729586.

13 M. Mitchell, Genetic Algorithms: An Overview. Complexity, vol. 1, pp. 31-39, 1995.

14 Горбань А. Н. Нейроинформатика / А. Н. Горбань и др. // Красноярск: СО РАН. 1999. – 564 с.

15 Chumachenko E. I. Features of hybrid neural networks use with input data of different types / Electronics and Control Systems, N 4(42) – Kyiv: NAU, 2014. – pp. 91–97.

16 Бодянский Е. В. Искусственные нейронные сети: архитектуры, обучение, применения / Е. В. Бодянский, О. Г. Руденко // Харьков: Телетех, 2004. – 369 с.

17 Bodyanskiy Ye. Hybrid wavelet-neuro-fuzzy system using adaptive W-neurons / Ye. Bodyanskiy, I. Pliss, O. Vynokurova // Wissenschaftliche Berichte, FH Zittau/Goerlitz. – no.106 (N.2454–2490). – 2010. – pp. 301-308.

18 Винокурова Е.А. Об одном алгоритме обучения адаптивной вэйвлет-нейронной сети на скользящем окне / Е.А. Винокурова, Н.С. Ламонова // Автоматизація виробничих процесів. – №21(2). – 2005. – С. 71–75.

19 Винокурова О.А. Функціонально-зв'язана багатовимірна вейвлет-нейро фаззі система для обробки хаотичних часових рядів. Комп'ютерні технології. – Вип. 130. – Т. 143. – 2010. – С. 71–76.

20 T. Kohonen and T. Honkela, “Kohonen network,” 2007, accessed: March 2012. [Online]. Available: http://www.scholarpedia.org/article/Kohonen network.

21 Kingma DP, Welling M. Auto-encoding variational bayes. arXiv preprint. 2013. arXiv:1312.6114

22 Leif E. Peterson (2009) K-nearest neighbor. Scholarpedia, 4(2):1883.

23 Greff K, et al. LSTM: a search space odyssey. IEEE transactions on neural networksand learning systems 2017;28(10):2222–32.

24 Chung J, Gulcehre C, Cho K, Bengio Y. Empirical evaluation of gated recurrent neural networks on sequence mod-eling. Eprint Arxiv. 2014. http://arxiv .org/abs/1412.3555v 1.

25 Cho, K. et al. Learning phrase representations using RNN encoder-decoder for statistical machine translation. arXiv preprint arXiv:1406.1078 (2014)

26 Breiman, L. Bagging Predictors. Machine Learning 24, 123–140 (1996). https://doi.org/10.1023/A:1018054314350

27 Y. Freund, and R. Schapire, “A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting”, 1997.

28 Wolpert, David H. “Stacked generalization.” Neural networks 5.2 (1992): 241-259.

29 ICU Survival Prediction using Ensemble Learning (Stacking) [Електронний ресурс] – Режим доступу до ресурсу: https://mc.ai/icu-survival-prediction-using-ensemble-learning-stacking/

30 Breiman, L. Random Forests. Machine Learning 45, 5–32 (2001). https://doi.org/10.1023/A:1010933404324

31 Verikas, A.; Vaiciukynas, E.; Gelzinis, A.; Parker, J.; Olsson, M. Electromyographic patterns during golfswing: Activation sequence profiling and prediction of shot effectiveness.Sensors 2016,16, 592

32 Friedman JH. Greedy function approximation: a gradient boosting machine. Ann Stat 2001;29(5):1189-1232. [doi:10.1214/aos/1013203451]

33 Ivakhnenko, A.G.. (1968). The Group Method of Data Handling – A rival of the Method of Stochastic Approximation. Soviet Automatic Control., vol. 1, pp. 43-55.

34 O. I. Chumachenko and K. D. Riazanovskiy, “Structural-parametric syntesis of neural network ensemble based on the estimation of individual contribution,” in Electronics and Control Systems, No 59, pp. 66-77, 2019.

35 G. Bebis and M. Georgiopoulos, "Feed-forward neural networks," in IEEE Potentials, vol. 13, no. 4, pp. 27-31, Oct.-Nov. 1994, doi: 10.1109/45.329294.

36 Broomhead, D., & Lowe, D. (1988). Radial basis functions, multi-variable functional interpolation and adaptive networks, Complex Systems, 2, 321–355.

37 R. Hecht-Nielsen, “Counterpropagation networks,” Applied Optics, vol. 26, pp. 4979–4984, 1987.

38 Donald F.Specht, Probabilistic neural networks, at Neural Networks, vol.3, iss.1, 1990, pp.109-118.

39 Nauck, Detlef & Kruse, Rudolf. (1995). NEFCLASS - a neuro-fuzzy approach for the classification of data. Proceedings of the ACM Symposium on Applied Computing. 461-465. 10.1145/315891.316068.

40 Lewis, D. (1998) Naive Bayes at forty: the independence assumption in information retrieval. In Machine Learning: ECML-98, Proceedings of the 10th European Conference on Machine Learning, Chemnitz, Germany (pp. 4–15). Berlin: Springer.

41 J. R. Quinlan. 1986. Induction of Decision Trees. Mach. Learn. 1, 1 (March 1986), 81–106. DOI:https://doi.org/10.1023/A:1022643204877

42 Platt, J., Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In: Advances in Large Margin Classifiers, MIT Press, Cambridge, MA, 1999

43 Tensorflow library [Електронний ресурс] – Режим доступу до ресурсу: https://www.tensorflow.org/

44 Keras library [Електронний ресурс] – Режим доступу до ресурсу: https://keras.io/

45 PyTorch library [Електронний ресурс] – Режим доступу до ресурсу: https://pytorch.org/

46 SciKit-Learn library [Електронний ресурс] – Режим доступу до ресурсу: https://scikit-learn.org

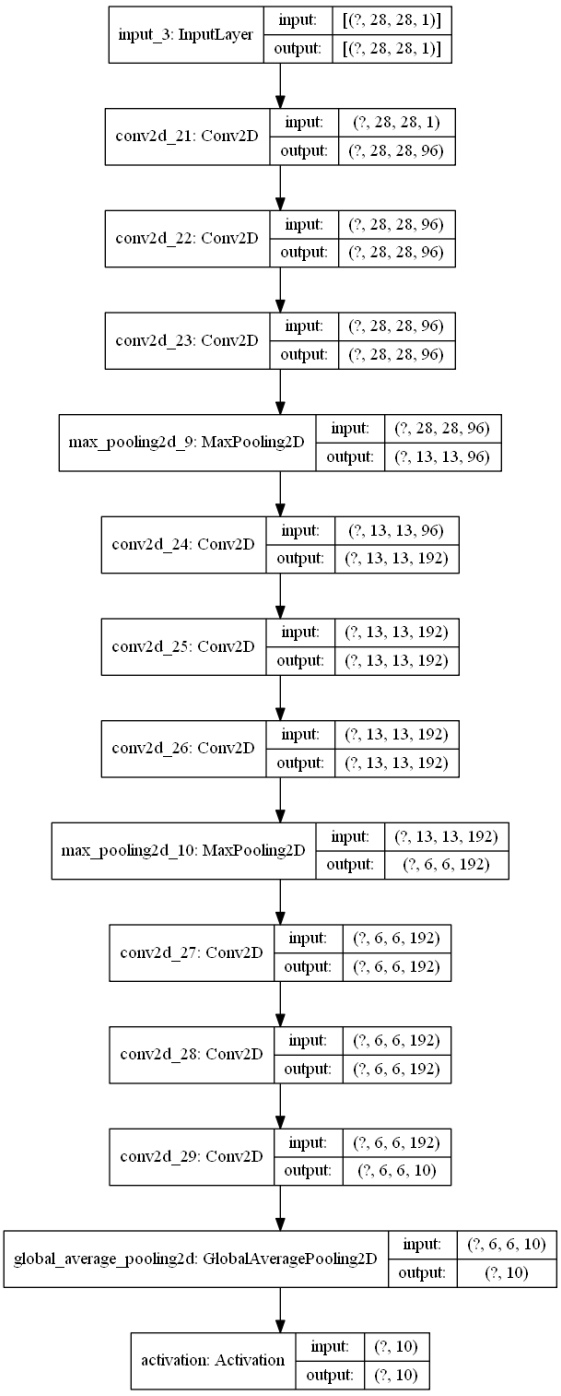
ВІДОМІСТЬ ДИПЛОМНОГО ПРОЄКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ІТ-62.24.1081.01 ПЗ	Пояснювальна записка	94	
3	А4	ІТ-62.10.1081.02 ТП	Додаток А. Відомість дипломного проєкту	1	
4	А4		Додаток Б. Результати перевірки на співпадіння	1	
5	А4	ІТ-62.10.1081.03 ВЗ	Додаток В. Топологія моделі ConvPool-CNN-C	1	
6	А4	ІТ-62.10.1081.04 ВЗ	Додаток Г. Топологія моделі ALL-CNN-C	1	
7	А4	ІТ-62.10.1081.05 ВЗ	Додаток Ґ. Топологія моделі Network in Network CNN	1	
8	А4	ІТ-62.10.1081.06	Додаток Д. Програмний код єдиного інтерфейсу	1	
9	А4	ІТ-62.10.1081.07	Додаток Е. Програмний код радіально-базисної мережі	1	
10	А3	ІТ-62.10.1081.08-11 ВЗ	Додатки креслення	4	

					ІТ-62.24.1081.02 ТП		
		ПІБ	Підп.	Дата			
Розробн.		Рязановський К.Д.			Відомість дипломного проєкту	Лист	Листів
Керівн.		Чумаченко О.І.				97	107
Консульт.		Пасько В.П.				КПІ ім. Ігоря Сікорського Каф. ТК Гр. ІТ-62	
Н/контр.		Лісовиченко О.І.					
Зав.каф.		Пархомей І.Р.					

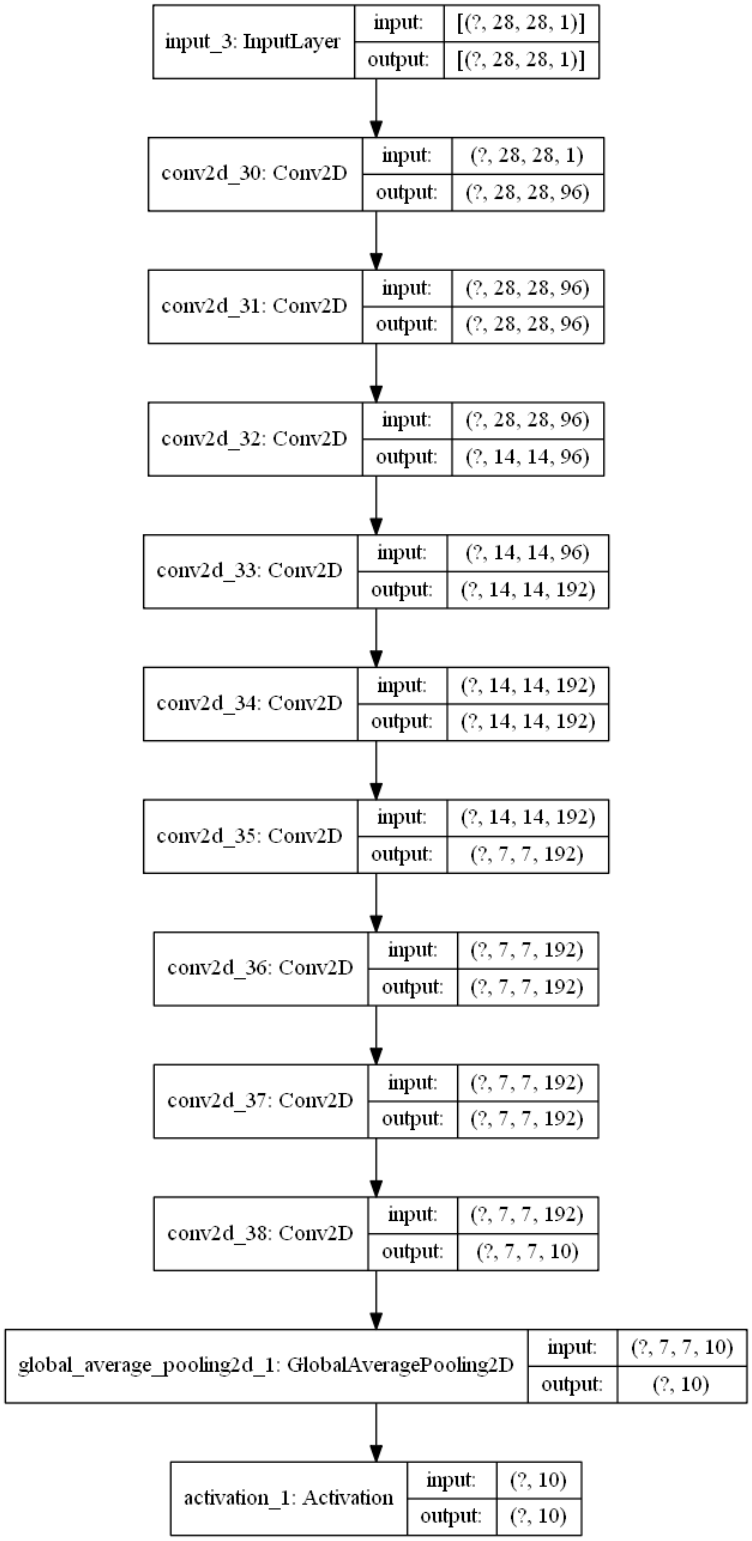
ДОДАТОК В

Топологія моделі ConvPool-CNN-C



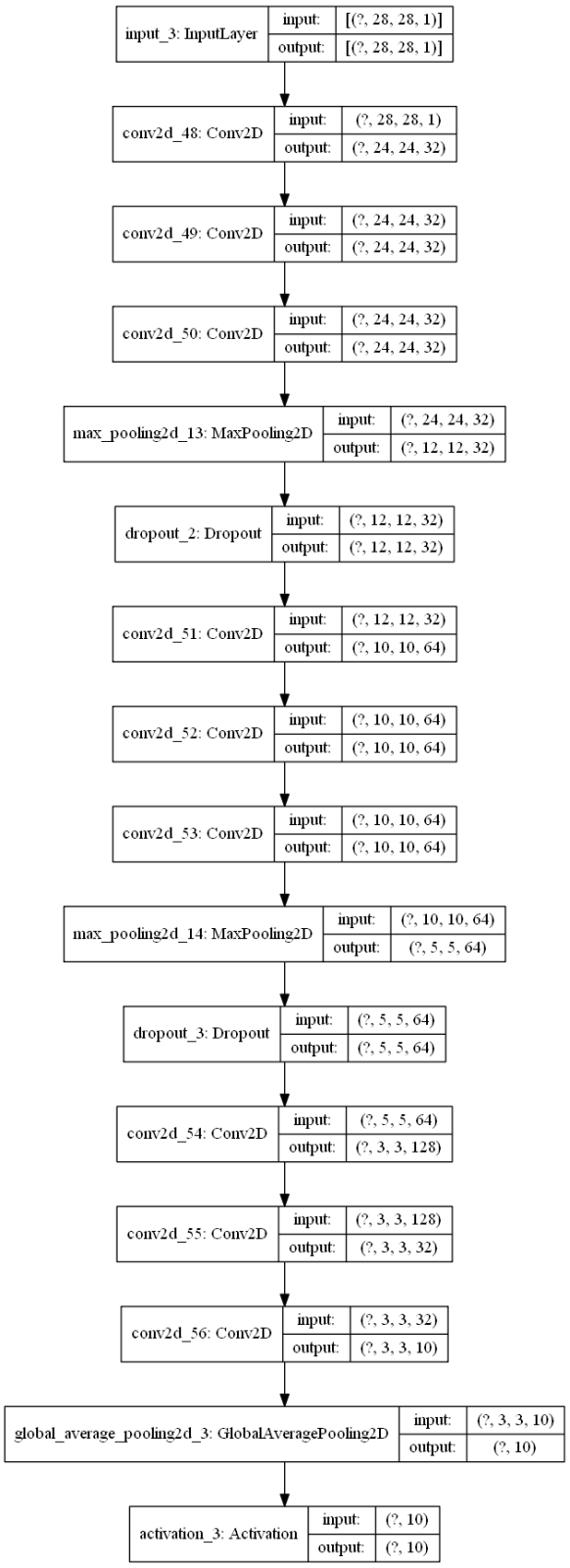
ДОДАТОК Г

Топологія моделі ALL-CNN-C



ДОДАТОК Ґ

Топологія моделі Network In Network CNN



ДОДАТОК Д

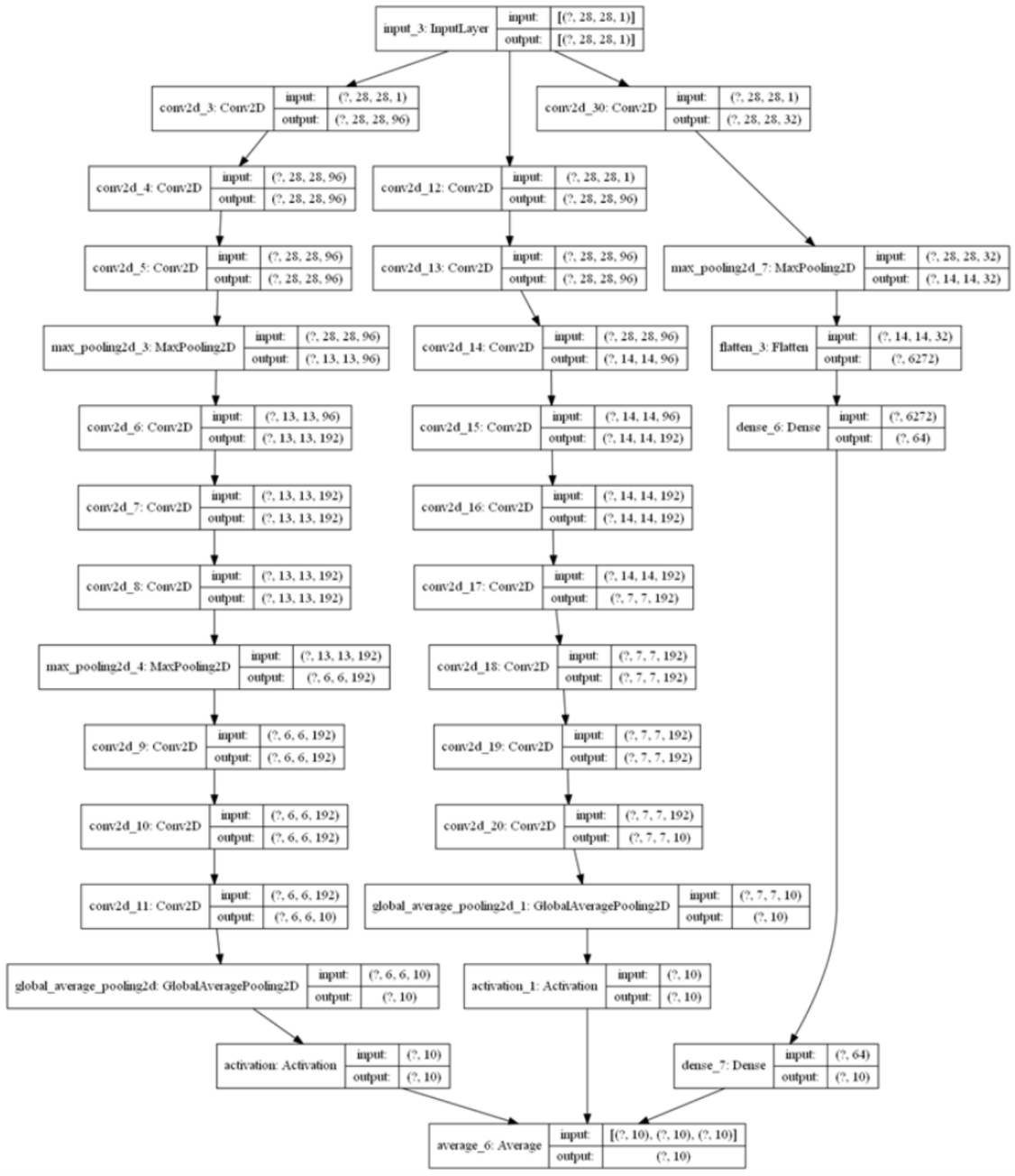
Програмний код єдиного інтерфейсу для розроблених нейронних мереж

```
class BaseNN:
    def _compile(self, optimizer, loss, **kwargs):
        """Used for Keras NN models that must be compiled before fitting"""
        pass

    def fit(self, X, y, **kwargs):
        """Training (fitting) model based on the given training data X and y"""
        raise NotImplementedError(
            "Must be implemented in application classes")

    def predict(self, X):
        """Make a prediction (class or continuos value) for the given data X"""
        raise NotImplementedError(
            "Must be implemented in application classes")

    def predict_proba(self, X):
        """Applicable only to classifiers.
        Return probabilities of belonging to each class for the given data X
        """
        pass

    def evaluate(self, X, y):
        """Determine classification score
        or approximation MAPE for the given data X and y
        """
        raise NotImplementedError(
            "Must be implemented in application classes")
```

ДОДАТОК Е

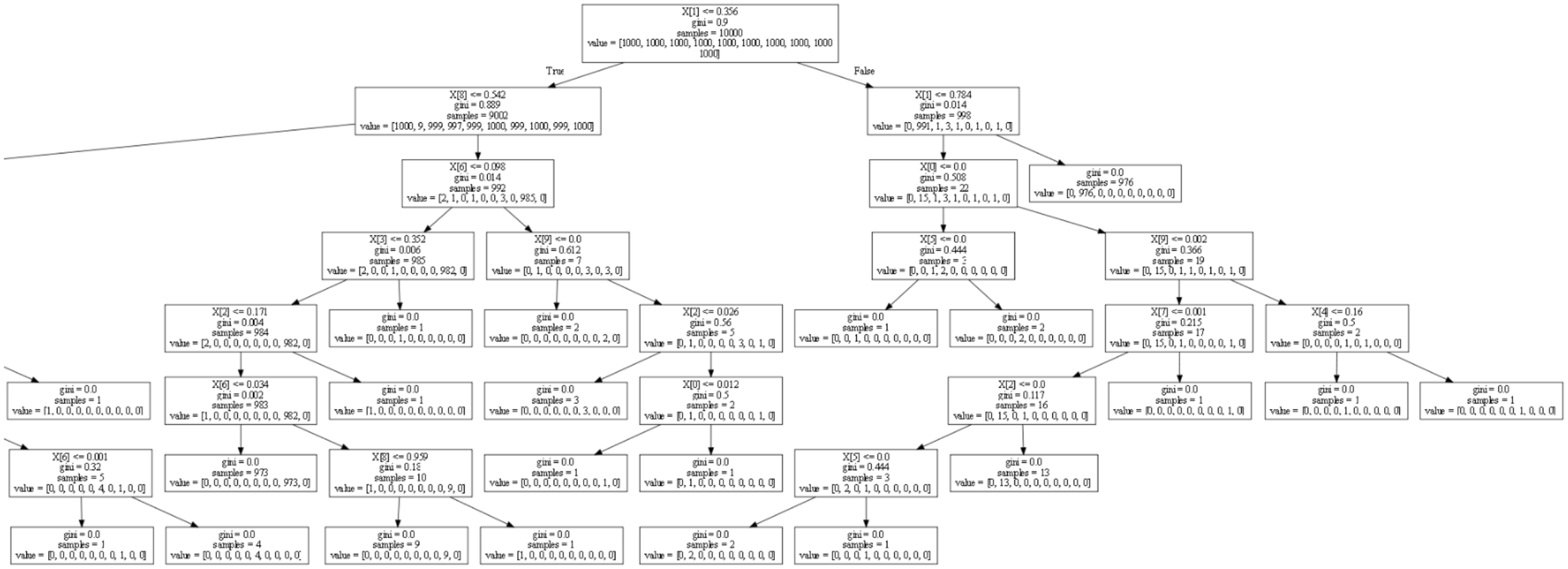
Програмний код радіально-базисної мережі в розробленому інтерфейсі

```
class RBFN(KerasModel):
    def __init__(self, n_clusters, input_shape, beta=0.3):
        self.n_clusters = n_clusters
        self.input_shape = input_shape
        self.beta = beta

    def _build(self, n_classes):
        rbfn = keras.Sequential()
        rbfn.add(RBFLayer(self.n_clusters,
                    initializer=self.initializer,
                    betas=self.beta,
                    input_shape=(self.input_shape,)))
        rbfn.add(Dense(n_classes, activation='softmax'))
        self.model = rbfn

    def fit(self, X, y, epochs=100, batch_size=20, optimizer='adam', loss='sparse_c
ategorical_crossentropy'):
        ss = StandardScaler()
        X = ss.fit_transform(X)
        self.ss = ss

        self.initializer = InitCentersKMeans(X, self.n_clusters)

        metric = 'accuracy'
        self._fit(X, y, optimizer, loss, metric, epochs, batch_size)

    def predict(self, X):
        X = self.ss.transform(X)
        return super().predict(X)

    def predict_proba(self, X):
        X = self.ss.transform(X)
        return super().predict_proba(X)

    def evaluate(self, X, y):
        X = self.ss.transform(X)
        return super().evaluate(X, y)
```



Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

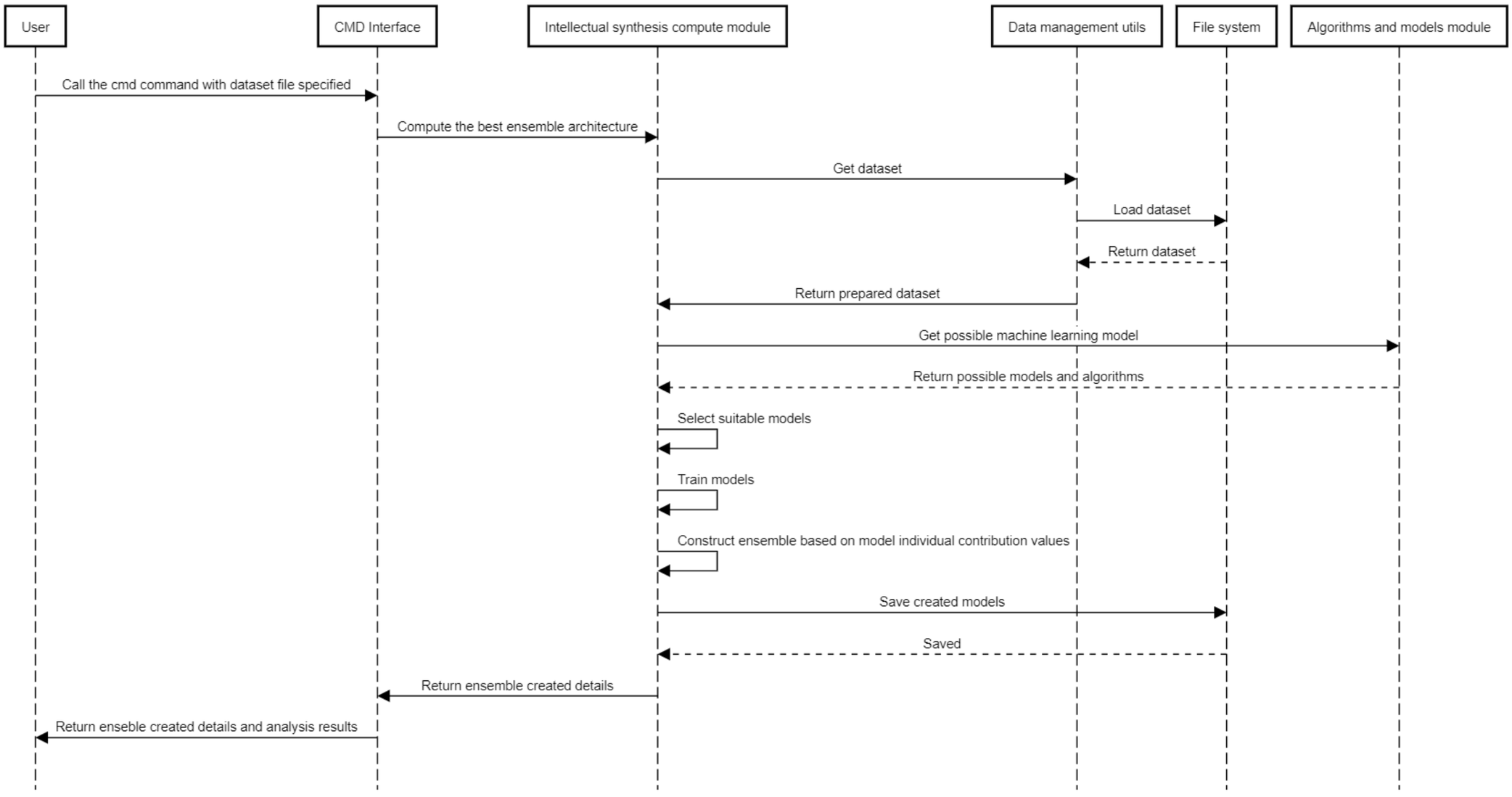
Зм.	Лист	№ докум	Підпис	Дата	IT.62.24.1081.08		
Розроб.		Рязановський			Додаток Є. Частина архітектури скороченого ансамблю з прикладу 2	Літ.	Маса / Мірило
Перев.		Чумаченко				Лист	Листів
					Кафедра Технічної кібернетики	Група IT-62	
Затв.		Пасько В. П.					



Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата	IT.62.24.1081.09		
Розроб.		Рязановський			Додаток Ж. Частина архітектури мета-дерева з прикладу 2	Літ.	Маса / Мірило
Перев.		Чумаченко				Лист	Листів
Затв.		Пасько В.П.			Кафедра Технічної кібернетики	Група IT-62	

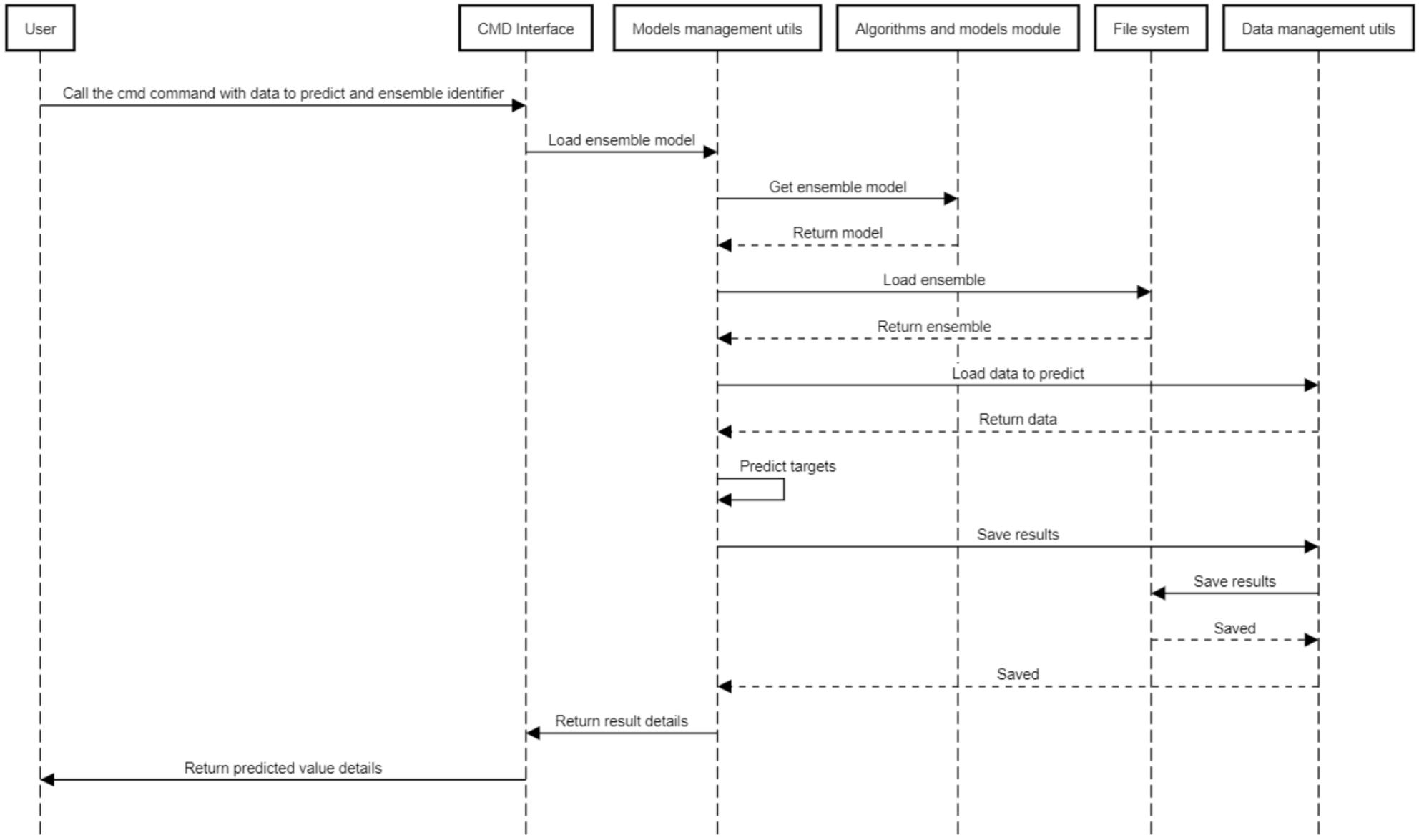
Ensemble creation workflow



Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

Зм.	Лист	№ докум	Підпис	Дата	IT.62.24.1081.10		
Розроб.		Рязановський			Додаток 3. Діаграма послідовності для створення ансамблю	Літ.	Маса / Мірило
Перев.		Чумаченко				Лист	Листів
Затв.		Пасько В.П.			Кафедра Технічної кібернетики	Група ІТ-62	

Prediction workflow



					IT.62.24.1081.11	
Зм.	Лист	№ докум	Підпис	Дата	Додаток И. Діаграма послідовності для прогнозування ансамблем	Літ. / Маса / Мірило
Розроб.		Рязановський				
Перев.		Чумаченко				Лист / Листів
					Кафедра Технічної кібернетики	Група ІТ-62
Затв.		Пасько В.П.				

Підпис і дата | Інв. № дубл. | Взам. інв. № | Підпис і дата | Інв. № ориг.

UNICHECK

Власник документу:
Лісовиченко Олег Іванович

Дата перевірки:
11.06.2020 11:49:58 EEST

Дата звіту:
11.06.2020 11:52:06 EEST

ID перевірки:
1003953143

Тип перевірки:
Doc vs Internet + Library

ID користувача:
76913

Назва документу: **IT-62_Рязановский**

ID файлу: **1003968196** **Кількість сторінок:** **70** **Кількість слів:** **13450** **Кількість символів:** **100884** **Розмір файлу:** **727.68 KB**

8.14% Схожість

Найбільша схожість: **1.85%** **з джерело бібліотеки.** **ID файлу:** **1000045688**

1.3% Схожість з Інтернет джерелами	21	Page 72
7.86% Текстові збіги по Бібліотеці акаунту	64	Page 72

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Заміна символів	8